

# e-book

# Curso: Front-End do Bem

**100% Gratuito | Sem pegadinhas | Sem pedir e-mail**

- Inicio do Curso
- Instalação do MS Visual Studio Code
  - HTML 5
  - CSS 3
  - Git
  - GitHub
- Sugestões de Prática no final de cada tema abordado



/igor-rebolla

# Conteúdo:

## ***Inicio do Curso***

### ***Instalação do MS Visual Studio Code***

- Estrutura de Pastas dentro do MS Visual Studio Code
- Sugestões de Prática para esse tema

### ***1º Pilar do Curso (HTML)***

- O que é HTML?
- Estrutura Básica do Documento HTML
- Explicação de Cada Elemento(Tag) da Estrutura Básica do Documento HTML
- Sugestões de Prática para esse tema
- Conceito Adicional e Exemplo
- Semântica
- Como a "Anatomia" dessas(Tags) são formadas
- Exemplo usando 4 Tags no mesmo documento HTML
- Sugestões de Prática para esse tema
- O que são Atributos em uma TAG HTML
- Tag <img> e seus atributos
- Tag <a> e seu atributo
- Exemplo usando 7 Tags no mesmo documento HTML
- Sugestões de Prática para esse tema



# Conteúdo:

- Tag <div>
- O que é um id ?
- O que é uma classe(class) ?
- Dicionário com os Termos usados até essa aula
- Sugestões de Prática para esse tema

## **2º Pilar do Curso (CSS)**

- O que é CSS ?
- Como a Anatomia(Estrutura) do CSS é formada?
- Como a comunicação é feita entre o HTML e o CSS?
- Sugestões de Prática para esse tema
- O que é uma Box Model ?
- Exemplos sobre Box Model
- Eu sou uma Box Model
- Sugestões de Prática para esse tema
- Padding e Margin (Reforçando)
- Exemplos Padding e Margin
- Sugestões de Prática para esse tema
- O que é FlexBox?
- Propriedades do FlexBox
- Exemplos de FlexBox
- Sugestões de Prática para esse tema



# Conteúdo:

- O que é CSS Grid?
- Propriedades do CSS Grid
- Exemplos de CSS Grid
- Sugestões de Prática para esse tema

## ***Conteúdo Extra:***

### ***O que é o Git?***

- Instalação do Git

### ***O que é o GitHub?***

- Instalação do GitHub
- Exemplo Prático do GitHub
- Sugestões de Prática para esse tema



/igor-rebolla

# Inicio do Curso

## Curso: Desenvolvedor(a) Front-End para quem está querendo migrar de área (Front-End do Bem)

O Curso foi criado depois de conversar com 35 profissionais das áreas (Administrativa, Engenharia, Financeiro, Jurídico...) e essas pessoas relataram que tem vontade de migrar para a Área de Tecnologia (Front- End) só que acham os termos complicados.

Esse curso vai procurar tratar os “termos complicados” de forma lúdica (tanto que os exemplos serão baseados no desenvolvimento /construção de uma casa).

Vamos trabalhar com 3 pilares dentro do curso para que vocês possam dar os Primeiros passos como Desenvolvedor(a) Front-End. Esses pilares são:

**HTML 5 / CSS 3 / *JavaScript***

Obs.: Na criação desse e-book (estamos na aula 12 do curso) contemplando o: HTML 5 e o CSS 3, por esse motivo JavaScript está escrito em vermelho acima (ainda não vimos essa parte).



# Inicio do Curso

Como assim Igor, qual a relação que uma CASA tem com o desenvolvimento FrontEnd? Eu respondo, todos os conceitos básicos abordados aqui, poderão ser replicados para a construção de um Site. O Exemplo abaixo mostra o que cada um dos 3 pilares do Front-End tem em relação com uma CASA:

- HTML 5 (Os códigos representam a Estrutura da casa);
- CSS 3 (Os códigos trazem estilo, como: a Cor da parede);
- JavaScript (Os códigos trazem ações para dentro da casa, como: Acender e Apagar uma lâmpada através do interruptor)

Antes de começarmos a mostrar os conceitos, uma casa para ser construída precisa de um Software (Programa) específico para dar formas, cores e etc com base no que o Cliente, Engenheiro, Arquiteto definiram e assim tornar visível aquela ideia (rascunho) que está no papel. O programa ao qual eu me refiro aqui é o AUTOCAD® (Este não será utilizado aqui no curso, apenas foi citado como exemplo).



# Inicio do Curso

E aí vocês me perguntam qual o programa que utilizaremos para o desenvolvimento de nossa CASA aqui no Curso? Aqui vamos utilizar 2 programas:

- Microsoft Visual Studio Code ( o qual vamos inserir nossos códigos);
- Google Chrome ( para visualizar o resultado dos códigos que fomos inserindo no Microsoft Visual Studio Code)

OBS.: Tanto o Microsoft Visual Studio Code quanto o Google Chrome são gratuitos.

"Tá você, falou, falou e aonde eu procuro esse tal de Microsoft Visual Studio Code para fazer a instalação?"

A partir da próxima página daremos inicio ao Procedimento de instalação do Microsoft Visual Studio Code.

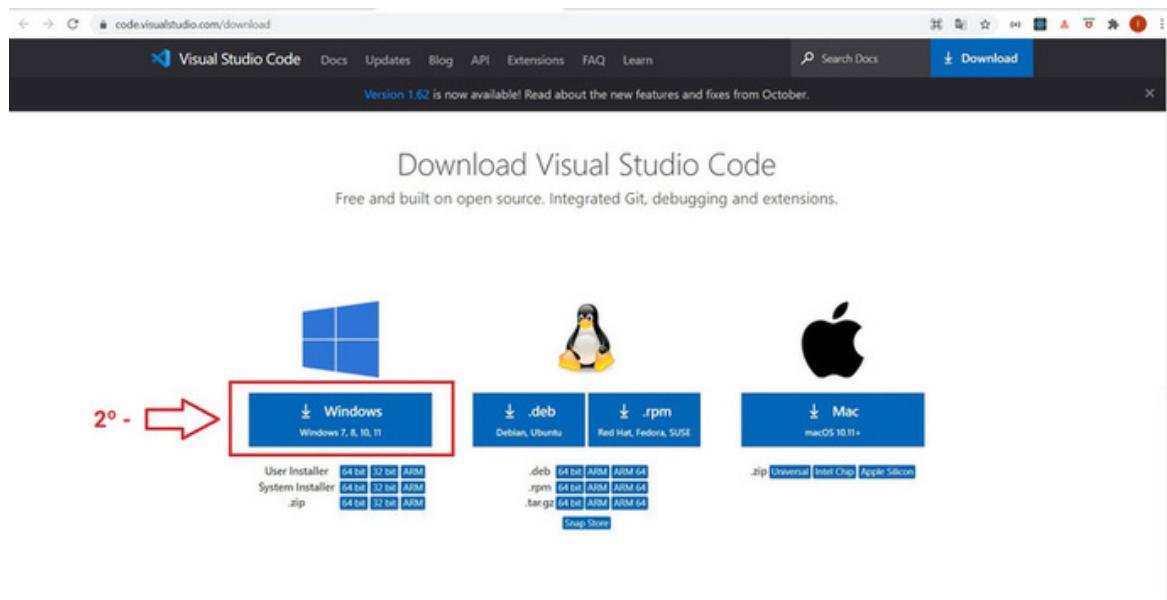


# Instalação do Microsoft Visual Studio Code

- o 1 - Clicar no link abaixo que vai dar acesso a página para baixarmos o Visual Studio (Aqui vamos usar a versão para Windows) que é o Sistema Operacional instalado aqui no meu computador.

[https://code.visualstudio.com/download \(Passo 1\)](https://code.visualstudio.com/download)

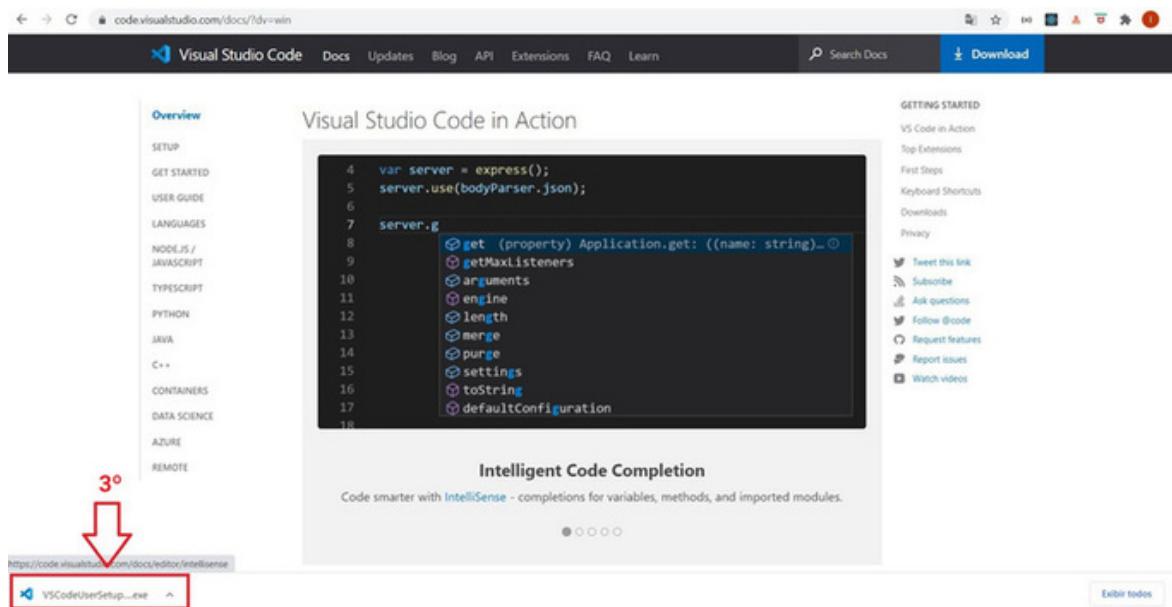
- 2° - Depois de clicar no Link (Passo 1), vai aparecer essa tela para vocês, vamos clicar na opção selecionada dentro do retângulo vermelho (Windows)



/igor-rebolla

# Instalação do Microsoft Visual Studio Code

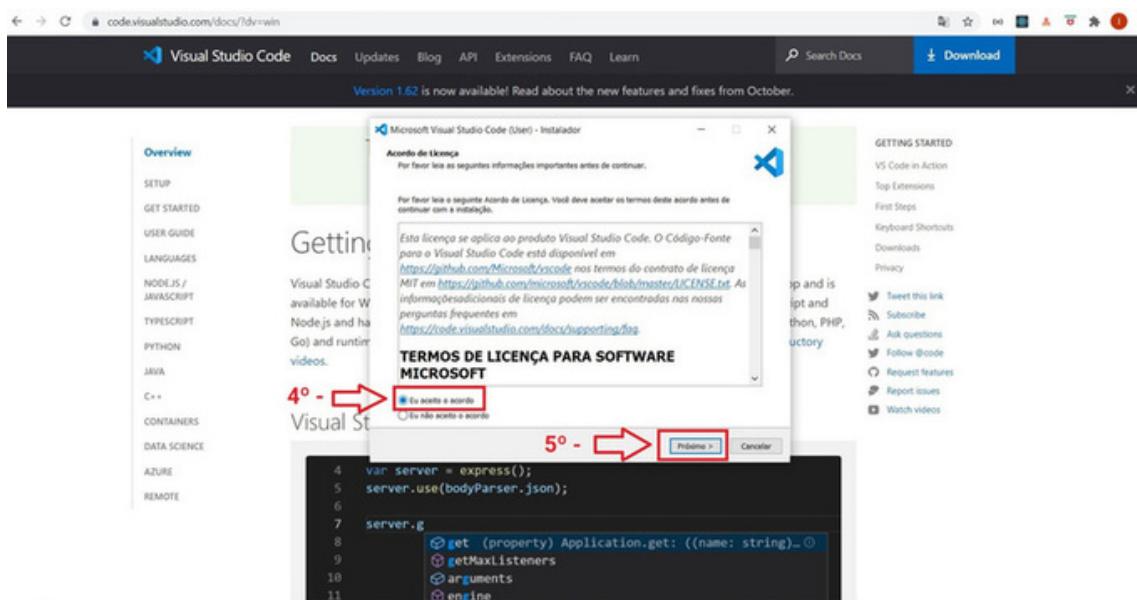
- 3º – Ao clicar na indicação do (Passo 2), o processo de download (baixar) o Microsoft Visual Studio Code começa automaticamente, só aguardar (dependendo da velocidade da conexão) pode demorar um pouquinho mais. Download concluído o arquivo para instalação ficará localizado no canto inferior esquerda da tela, conforme (Passo 3) abaixo:



/igor-rebolla

# Instalação do Microsoft Visual Studio Code

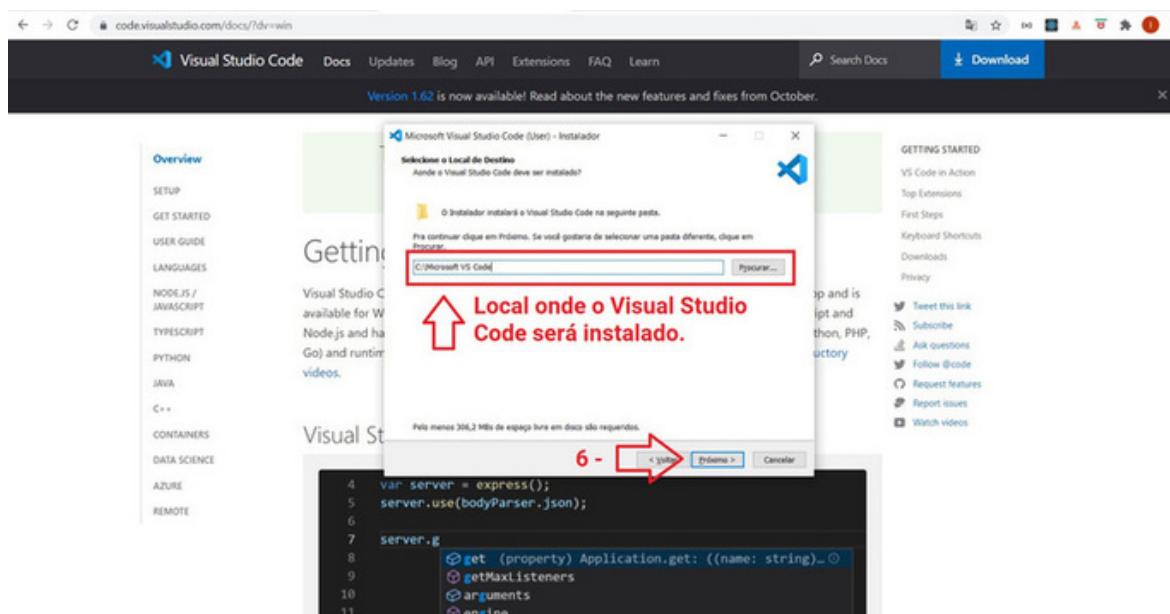
o 4º - Download efetuado, clicar 2x sobre o (Passo 3) que a instalação do Microsoft Visual Studio Code iniciará. Clicar em Eu aceito o contrato (Passo 4) e logo em seguida clicar em Próximo (Passo 5) conforme tela abaixo:



/igor-rebolla

# Instalação do Microsoft Visual Studio Code

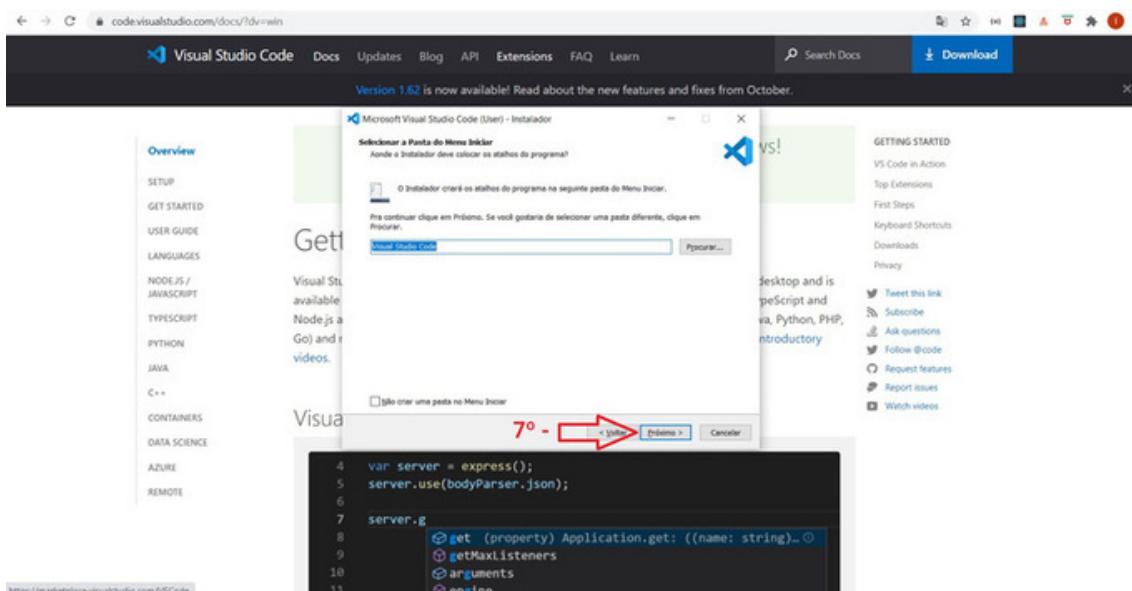
- 5º Depois de clicar em próximo conforme (Passo 5), a instalação do Microsoft Visual Studio Code, irá para a tela abaixo mostrando o local onde o Microsoft Visual Studio Code será instalado no seu computador. Minha sugestão aqui é não alterar o caminho que vem por padrão. Só clicar no botão próximo (Passo 6).



/igor-rebolla

# Instalação do Microsoft Visual Studio Code

- 6º - Depois de clicar em próximo conforme (passo 6), a tela abaixo aparecerá sugerindo um nome de Pasta para continuarmos nossa instalação (Aqui a sugestão também é deixar o que foi recomendado pela instalação). Só clicar em próximo conforme (passo 7);

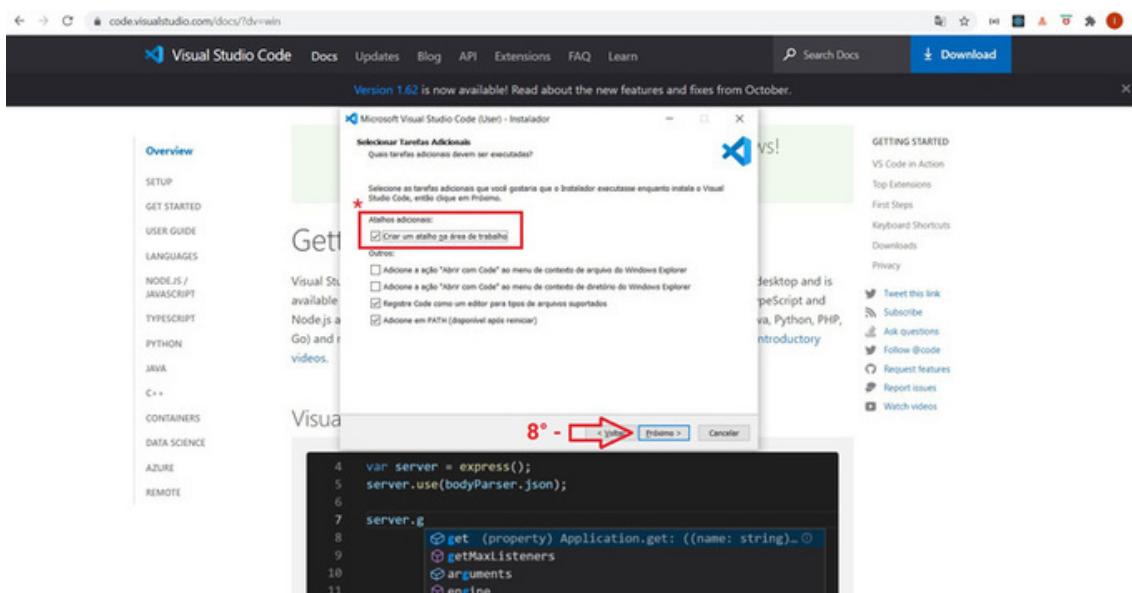


/igor-rebolla

# Instalação do Microsoft Visual Studio Code

o

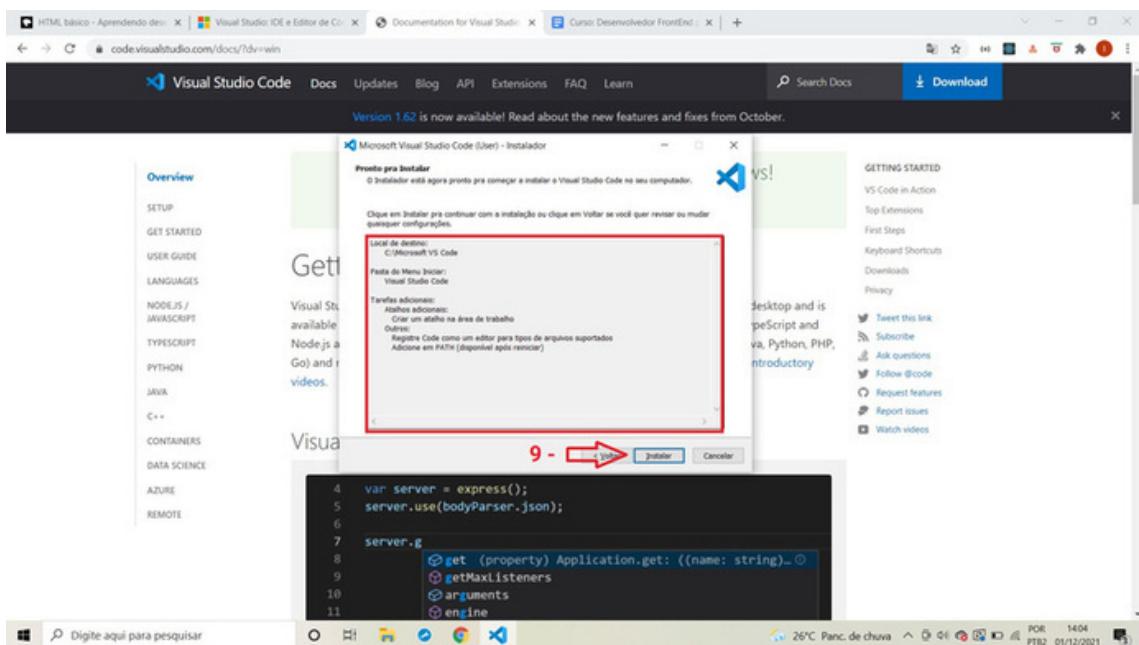
7° - Clicou em próximo conforme (passo 7), poderemos escolher se queremos ou não (colocar um atalho do Microsoft Visual Studio Code na área de trabalho) representado aqui por um \*, eu marquei para colocar pois facilita minha vida (lembrando que isso é opcional), as outras opções na tela eu sugiro não alterar. Só clicar em próximo.



/igor-rebolla

# Instalação do Microsoft Visual Studio Code

8º A tela abaixo mostra um resumo das opções selecionadas nas telas anteriores. Se estiver tudo OK, só clicar em instalar e aguardar a conclusão da instalação.

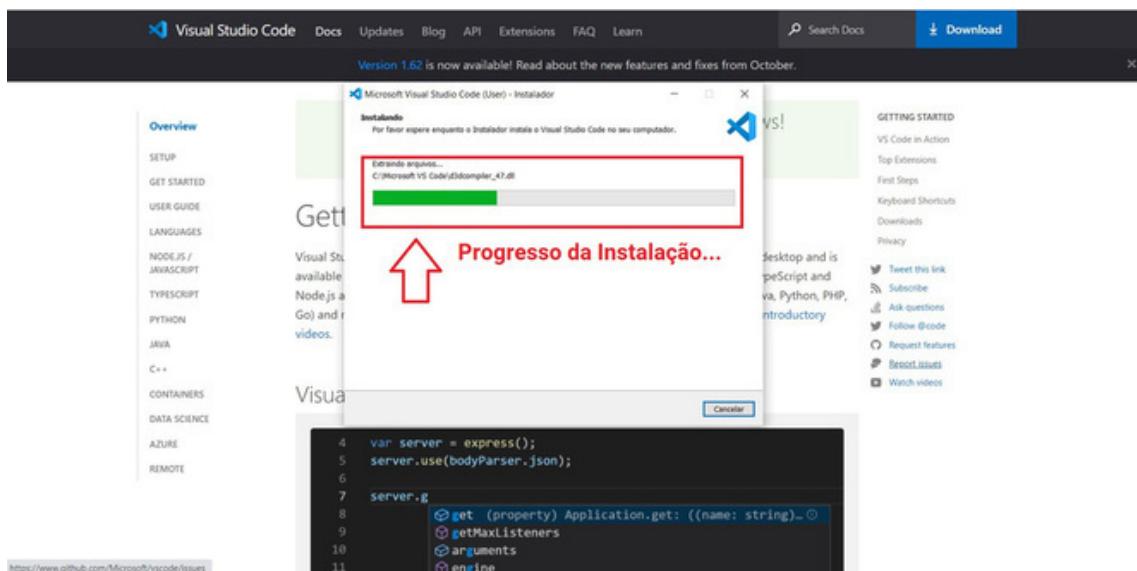


/igor-rebolla

# Instalação do Microsoft Visual Studio Code

o

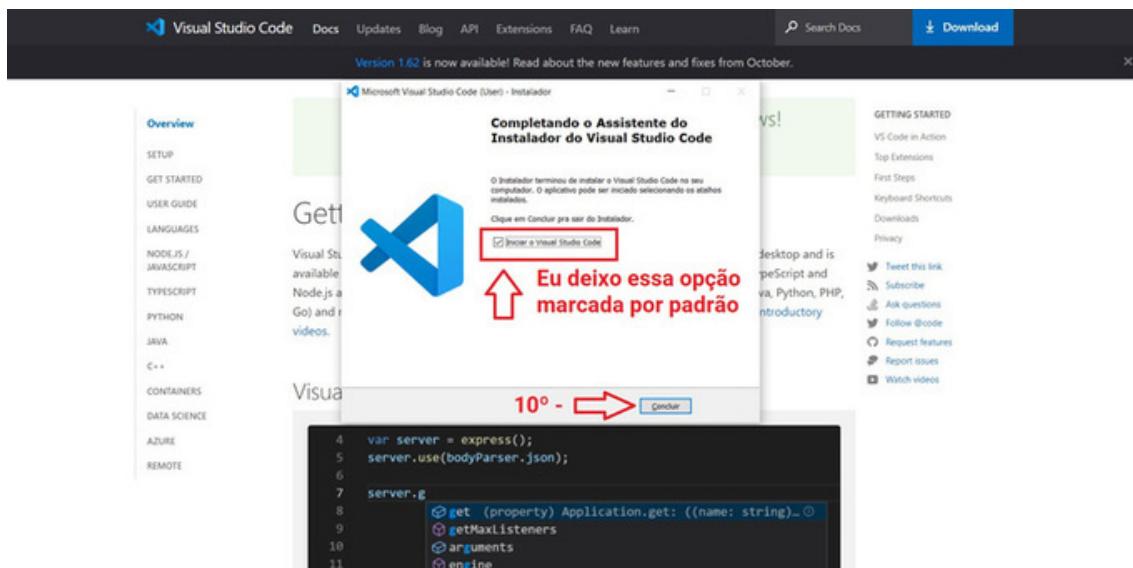
9º – Clicou em instalar conforme (passo 9) a barra de progresso aparecerá indicando o andamento da instalação:



/igor-rebolla

# Instalação do Microsoft Visual Studio Code

- 10º - Barra de progresso concluída, automaticamente ela vai para a tela abaixo, indicando que a instalação do Microsoft Visual Studio Code terminou e se você deseja iniciar o Microsoft Visual Studio Code (caso queira iniciá-lo agora) só deixar a opção (Iniciar Visual Studio Code) marcada e clicar em Concluir

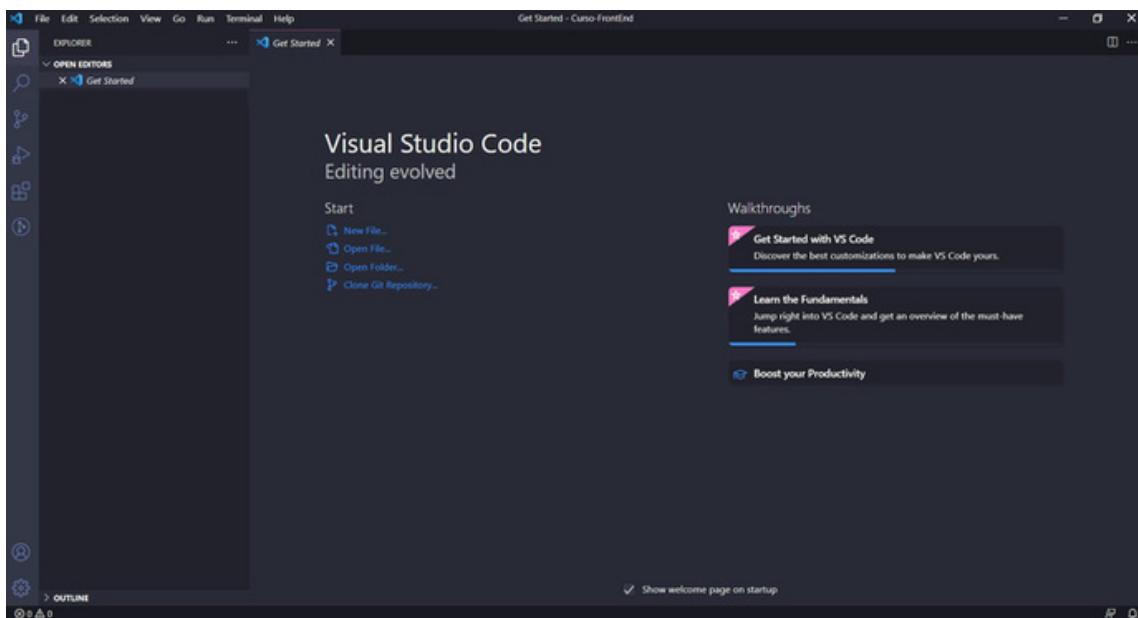


/igor-rebolla

# Instalação do Microsoft Visual Studio Code

Pronto (Microsoft Visual Studio Code) Instalado, abriremos o Programa para entender nesse primeiro momento como é a sua estrutura. A interface(tela) conforme figura abaixo que aparecerá ao iniciarmos o Microsoft Visual Studio Code pela primeira vez.

Pessoal... esse é o Microsoft Visual Studio Code, Microsoft Visual Studio Code esse é o Pessoal. Vamos nos ver muito nessa jornada que está apenas começando.



/igor-rebolla

# Estrutura de Pastas e Arquivos dentro do Microsoft Visual Studio Code

¶ - Agora que o Microsoft Visual Studio Code está instalado e vocês foram devidamente apresentados. Vamos ver como criar uma pasta (local onde os arquivos) do nosso curso vão ficar alocados e assim poderemos entender de forma visual como a estrutura de pastas funciona dentro do Microsoft Visual Studio Code.

Eu estou usando o Microsoft Windows 10, os passos aqui para criar uma pasta são:

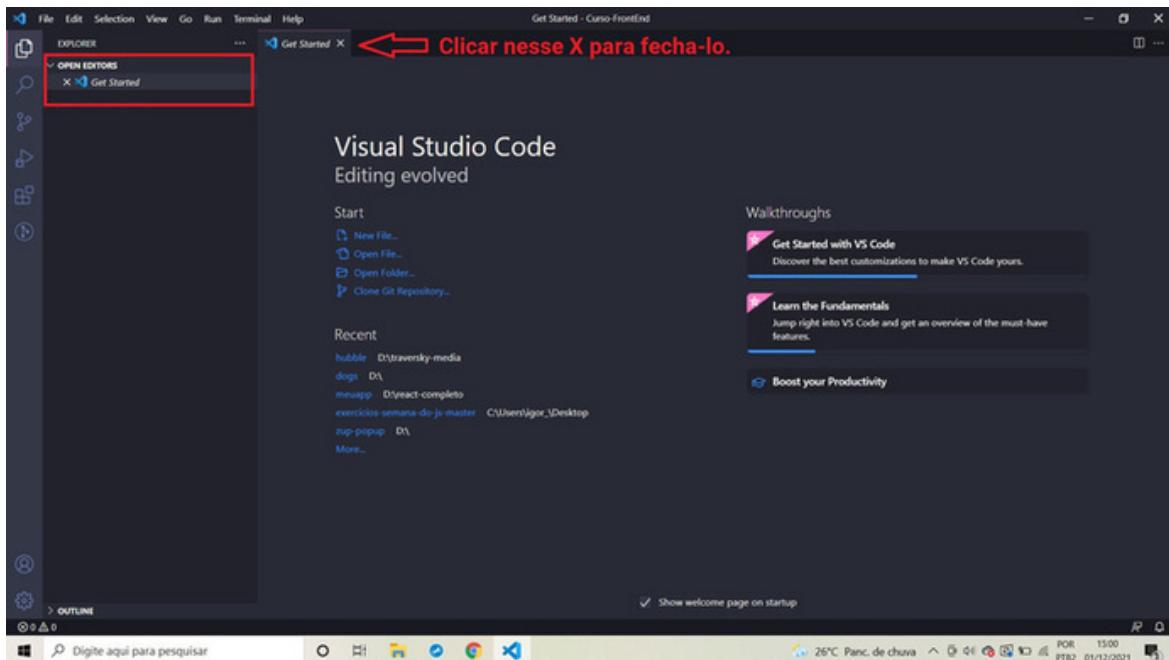
- Abrir o Explorador de Arquivos ou Windows Explorer;
- Selecionar o Local onde a pasta do curso será criada (eu escolhi D:/);
- Clicar com o botão direito do mouse e escolher a opção (novo e depois pasta);

Digitar o nome da Pasta - Aqui eu optei por Curso-FrontEnd



# Estrutura de Pastas e Arquivos dentro do Microsoft Visual Studio Code

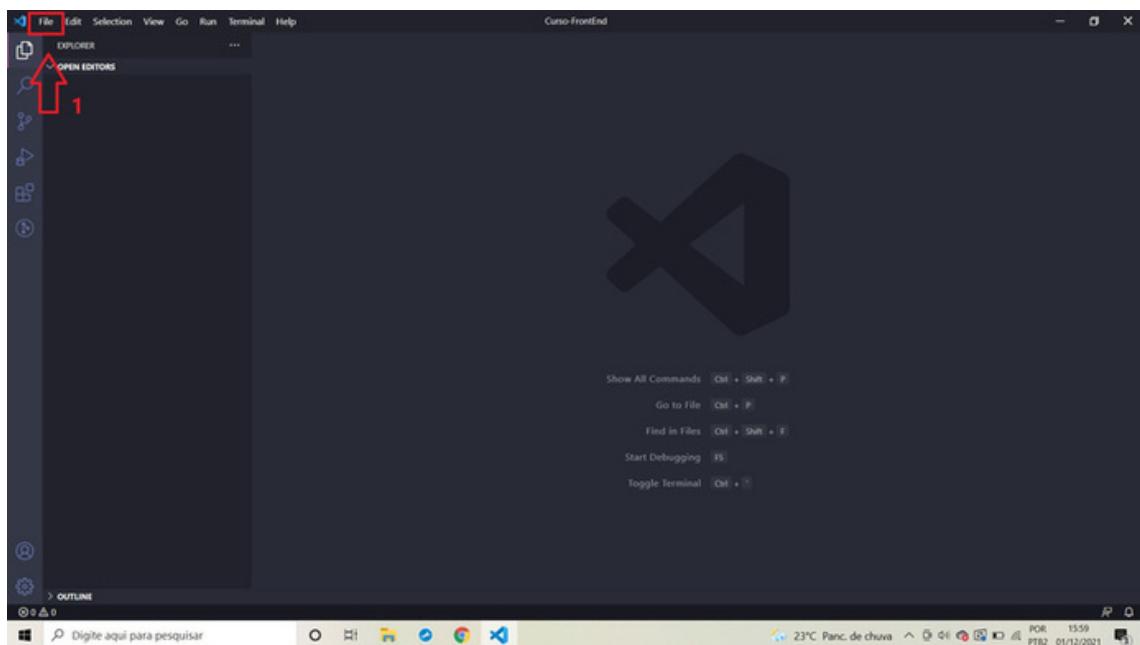
2º - Pasta Curso-FrontEnd devidamente criada, vamos abri-la dentro do Microsoft Visual Studio Code. Antes Vamos fechar o Get Started (para fazer isso basta clicar no X) conforme figura abaixo:



/igor-rebolla

# Estrutura de Pastas e Arquivos dentro do Microsoft Visual Studio Code

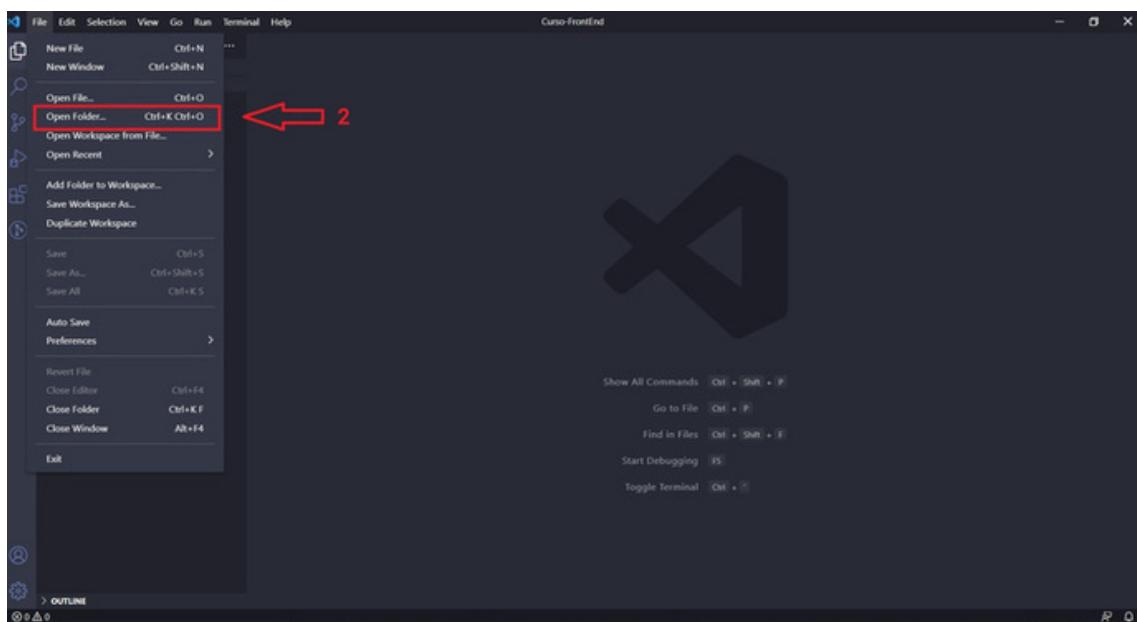
3º - Bora abrir dentro do Microsoft Visual Studio Code a pasta Curso-Front End. Para fazer isso, clicar em FILE (Passo 1)



/igor-rebolla

# Estrutura de Pastas e Arquivos dentro do Microsoft Visual Studio Code

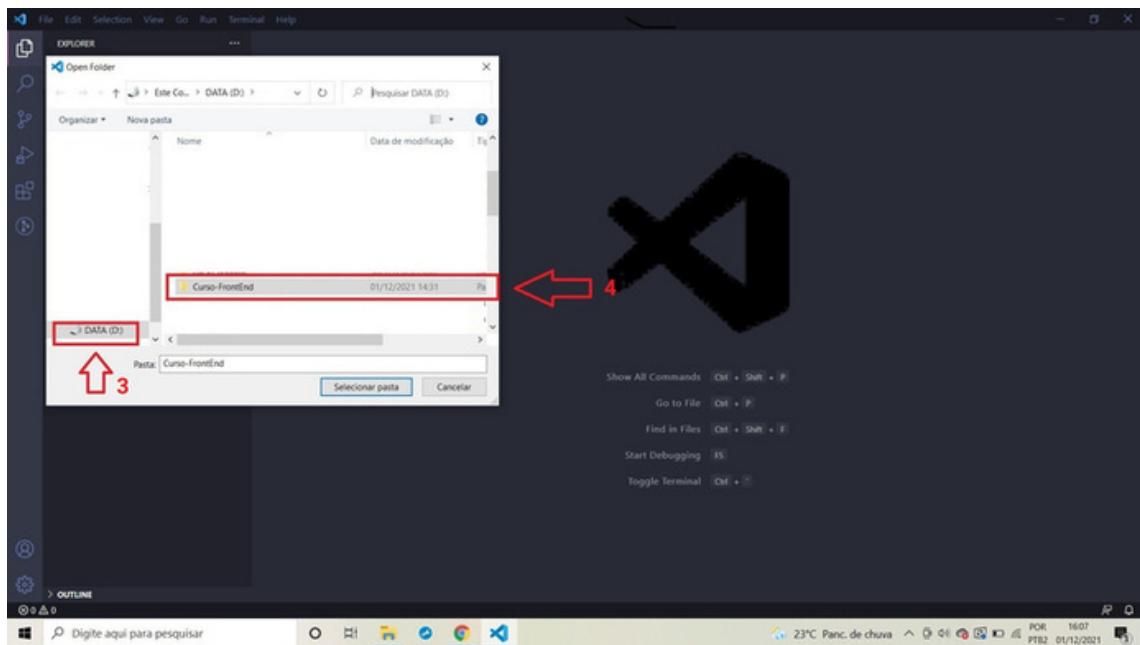
4º - E clicar em Open Folder (Passo 2)



/igor-rebolla

# Estrutura de Pastas e Arquivos dentro do Microsoft Visual Studio Code

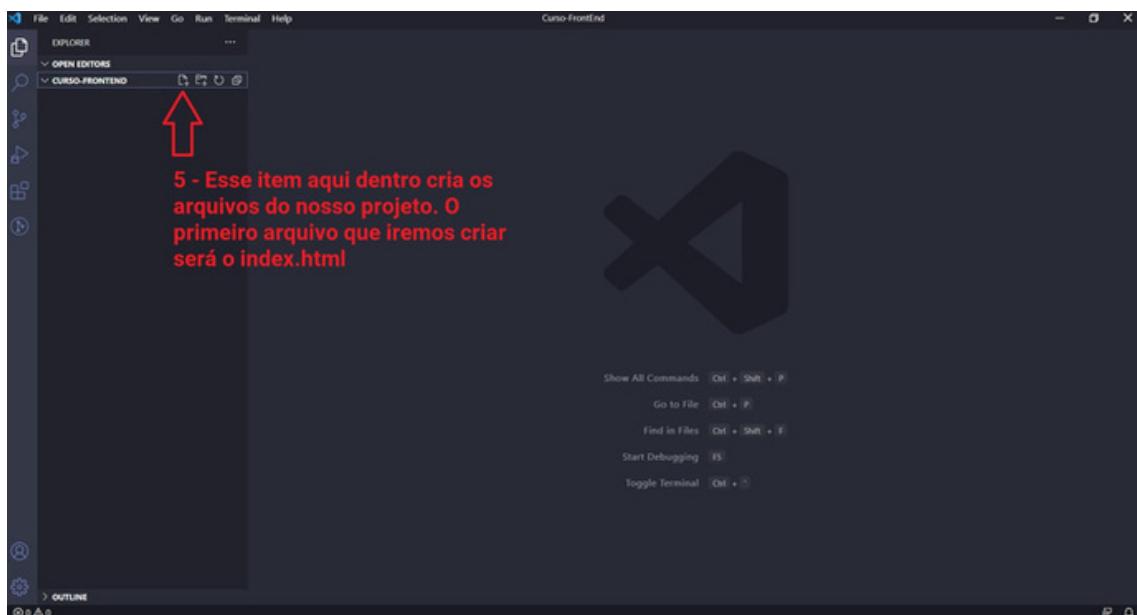
5º - Agora é clicar no local que criamos a pasta (Eu escolhi a letra D) conforme passo 3, localize a Pasta Curso-FrontEnd conforme passo 4 e clique 2x que a Pasta será aberta dentro do Microsoft Visual Studio Code.



/igor-rebolla

# Estrutura de Pastas e Arquivos dentro do Microsoft Visual Studio Code

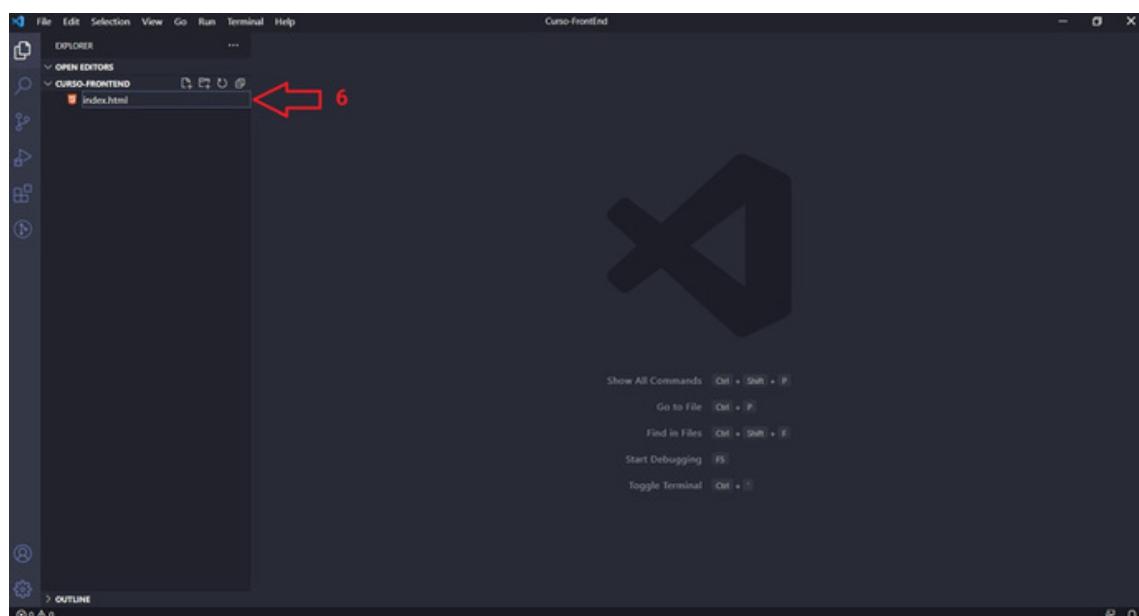
6º – Pronto a Pasta Curso-FrontEnd está aberta do lado esquerdo da tela (o lado esquerdo é o local onde o Microsoft Visual Studio Code disponibiliza toda a estrutura de pastas e arquivos que iremos utilizar em nosso projeto. Notem que não tem nada embaixo da pasta Curso-FrontEnd nesse momento. Vamos clicar no desenho do arquivo sinalizado pelo número 5 e criaremos o nosso primeiro arquivo do curso.



/igor-rebolla

# Estrutura de Pastas e Arquivos dentro do Microsoft Visual Studio Code

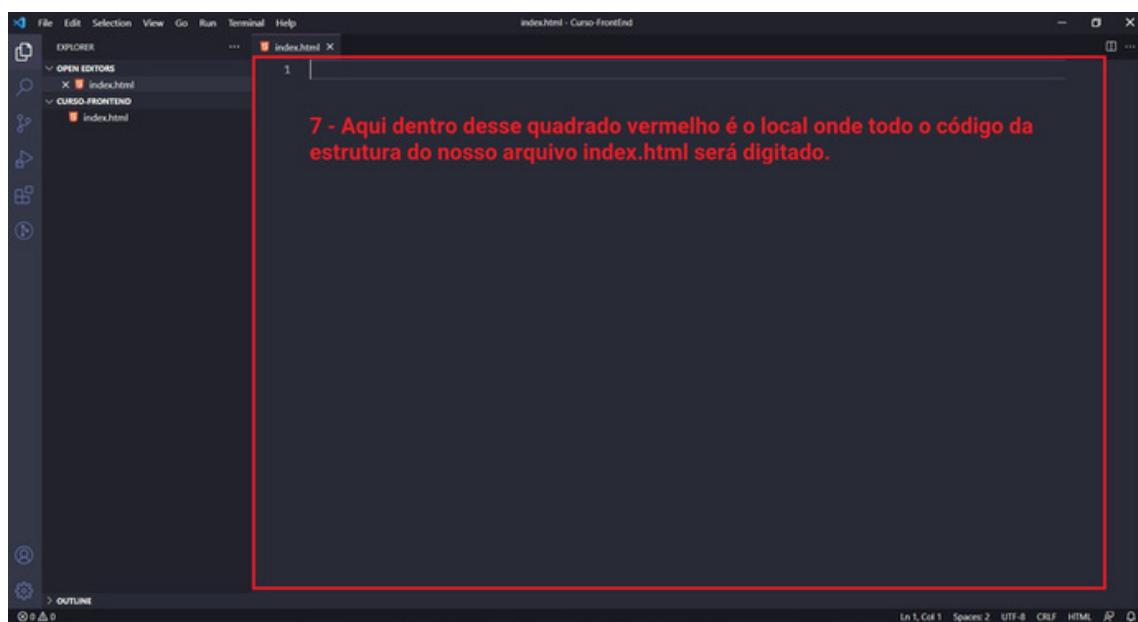
7º - Vamos nomear o primeiro arquivo do nosso curso, ao clicarmos no desenho do arquivo logo abaixo da pasta Curso-FrontEnd o cursor ficará piscando para ser digitado algo é aqui que daremos o nome de: **index.html** e logo em seguida pressionaremos o enter 1x para gravar esse nome.



/igor-rebolla

# Estrutura de Pastas e Arquivos dentro do Microsoft Visual Studio Code

8º – Notem que ao clicarmos sobre o index.html localizado do lado esquerdo da tela (logo abaixo da pasta Curso-FrontEnd) automaticamente do lado direito da tela, surgiu o mesmo arquivo index.html. O lado direito é o local onde vamos digitar todo o código que faz parte da estrutura desse arquivo index.html (conforme passo 7)



/igor-rebolla

# **Sugestões de prática para essa aula:**

- 1.Instalar o Microsoft Visual Studio Code (seguir os passos que foram mostrados);
- 2.Criar a pasta Curso-FrontEnd (Dentro do Explorador de Arquivos ou Windows Explorer no Windows);
- 3.Criar o arquivo index.html (Dentro do Microsoft Visual Studio Code, embaixo da pasta Curso-FrontEnd).



**/igor-rebolla**

# O que é HTML?

HTML (HyperText Markup Language) não é uma linguagem de programação, e sim uma linguagem de marcação que utilizamos dentro do Microsoft Visual Studio Code, programa esse que foi visto na aula 01 desse curso e o local onde vamos digitar os nossos códigos.

Código HTML devidamente digitado podemos ver o andamento do que estamos fazendo através do navegador (Google Chrome). Fazendo um paralelo com a construção de uma casa: Cada bloco que você vai colocando para levantar a estrutura de uma parede você vai vendo o andamento e essa parede vai tomando forma, é o que acontece exatamente quando você vai digitando os códigos.

Atualmente o HTML está na sua versão 5 (HTML 5).

Como o nosso exemplo aqui do Curso é baseado em entender a estrutura de uma Casa e o HTML é uma linguagem de marcação, podemos marcar os elementos HTML (estrutura) dentro do Microsoft Visual Studio Code que irá compor a nossa casa.

Aqui é só uma breve teoria do que é o HTML e como ele pode ser utilizado no decorrer do curso. Vamos entender mais com os exemplos, já nas próximas páginas ainda dentro dessa aula 02.

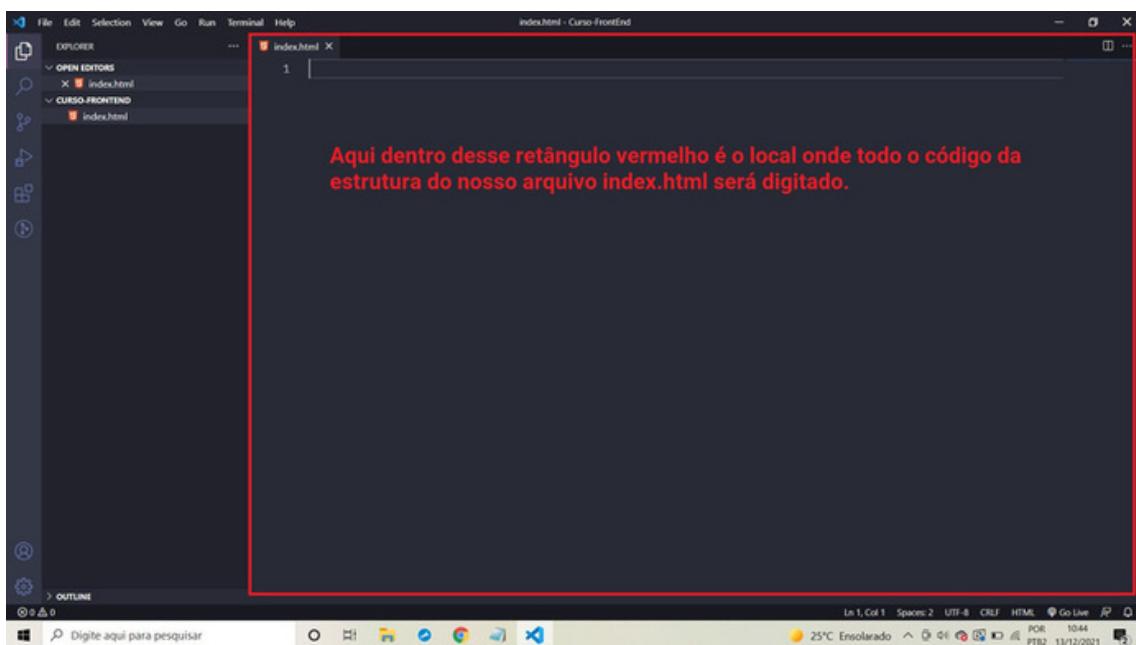


/igor-rebolla

# Estrutura Básica do Documento HTML

Na aula 01, uma das etapas foi criar a pasta Curso-FrontEnd e logo em seguida criamos dentro dessa pasta o nosso primeiro arquivo HTML o qual chamamos de index.html.

Pois bem, vamos abrir o Microsoft Visual Studio Code, depois a pasta Curso-Frontend e bora clicar em index.html. Irá aparecer para vocês uma tela conforme figura abaixo:



Será dentro desse espaço indicado pelo retângulo vermelho que iremos inserir (digitar) nossa Estrutura básica do Documento HTML. Que será mostrada logo no inicio da próxima página.

Preparados? Bora lá ver essa tal de estrutura...



# Estrutura Básica do Documento HTML

Essa é a Estrutura Básica do Documento HTML:

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>

<body>
</body>

</html>
```

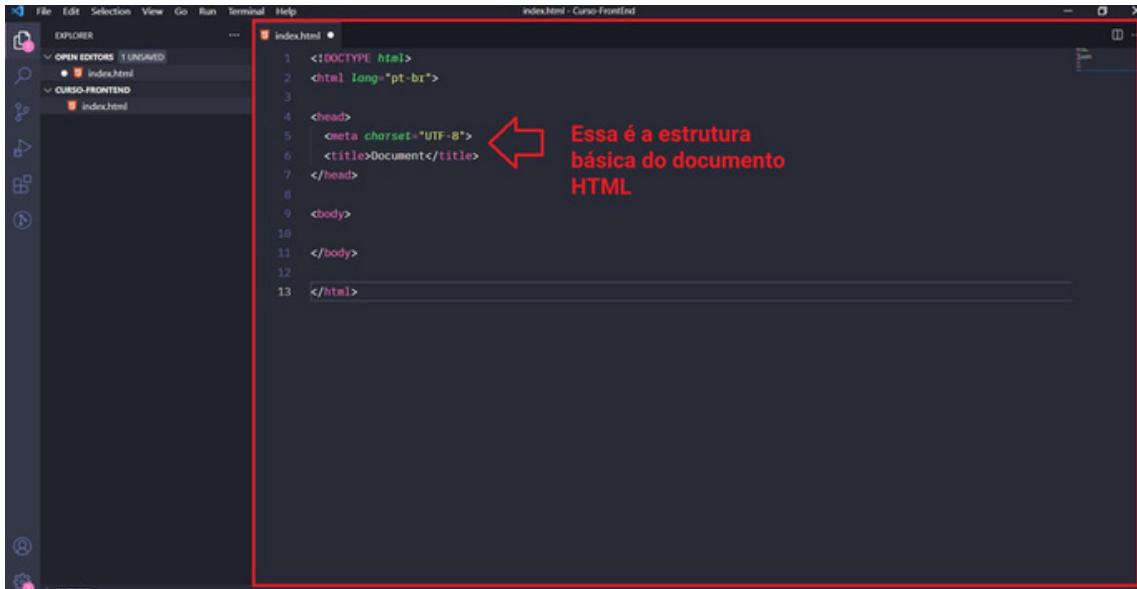


# Estrutura Básica do Documento HTML

1º - Agora que vimos qual é a estrutura básica do Documento HTML. Vamos inserir essa estrutura no Microsoft Visual Studio Code. Lembrando que antes disso, temos que seguir 3 passos:

- Abrir o Microsoft Visual Studio Code;
- Abrir a pasta Curso-FrontEnd;
- Abrir o arquivo index.html

A estrutura básica do Documento HTML ficará conforme imagem abaixo:



The screenshot shows the Microsoft Visual Studio Code interface. On the left is the Explorer sidebar with a file tree showing 'OPEN EDITORS' (empty) and 'CURSO-FRONTEND' (with 'index.html'). The main editor area has a red border and contains the following HTML code:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Document</title>
</head>
<body>
</body>
</html>
```

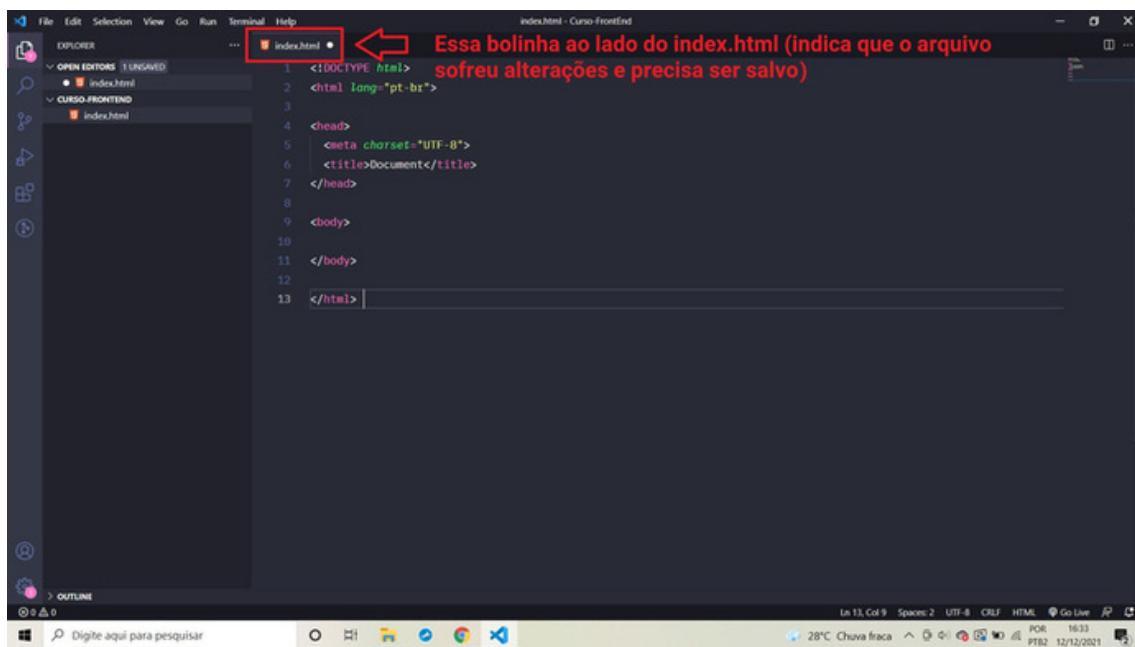
A red arrow points from the text 'Essa é a estrutura básica do documento HTML' to the opening line of the code, '<!DOCTYPE html>'.



/igor-rebolla

# Estrutura Básica do Documento HTML

Digitamos a Estrutura Básica do código HTML conforme figura da pagina anterior, agora precisamos salva-lo. Notem que ao lado de index.html tem uma **bolinha**, essa bolinha indica que o arquivo sofreu uma alteração e precisa ser salvo.



```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Document</title>
</head>
<body>
</body>
</html>
```



# Estrutura Básica do Documento HTML

Para salvar o arquivo index.html dentro do Microsoft Visual Studio Code, podemos fazer de 2 maneiras:

Clicar em File (Arquivo) e depois Save (Salvar);

No teclado (Pressionar a tecla Ctrl + tecla S) o sistema operacional que estou usando aqui é o Microsoft Windows 10.

Depois que o arquivo foi salvo (utilizando uma das maneiras acima) a bolinha que estava ao lado do arquivo index.html dará lugar ao X (indicando que o arquivo foi salvo). Conforme imagem abaixo:



# Explicação de Cada Elemento(Tag) da Estrutura Básica do Documento HTML

Depois que vimos nas páginas anteriores o que é o HTML e visão geral da estrutura básica do documento HTML, vamos entender o que cada uma daquelas 9 linhas digitadas dentro do Microsoft Visual Studio Code significam.

Ahhhhhhhhh, só um pequeno detalhe que não pode ser passado despercebido aqui. Notem que algumas linhas não se repetem e outras parecem ter uma ligação e se repetem acrescentando o / no inicio.

Bora entender sobre isso na próxima página....



# Explicação de Cada Elemento(Tag) da Estrutura Básica do Documento HTML

## **<!DOCTYPE html>**

A declaração aqui do DOCTYPE vai informar ao navegador, por exemplo o (Google Chrome) que a linguagem HTML está sendo utilizada.

## **<html lang="pt-br">**

Define a lang (language - linguagem/idioma) oficial do documento HTML que aqui foi definida para pt (português) - br(brasil).  
obs.: Notem que existe um html antes do lang = "pt-br", esse html faz referência ao </html> no final da estrutura básica que digitamos dentro do Microsoft Visual Studio Code.

## **<head>**

A Tag <head> é o local no qual iremos incluir Metadados, como o (meta charset="UTF-8") que veremos logo abaixo. E que precisamos incluir na página mas que não vai ser renderizado ou seja mostrado no navegador. Aqui funciona como o Impermeabilizante que usamos na parede para não pegar Umidade depois de rebocada e com a tinta aplicada, não vemos sua ação, mas ele está atuando nos bastidores e é importantíssimo.

## **<meta charset="UTF-8">**

Indica que a tabela de caracteres que vai ser utilizada pelo seu sistema será bem visualizado em qualquer lugar do mundo. E sem o UTF-8 não conseguimos digitar acentos (^, ~, ´ e etc) aparecendo uns "caracteres estranhos" em nosso código.



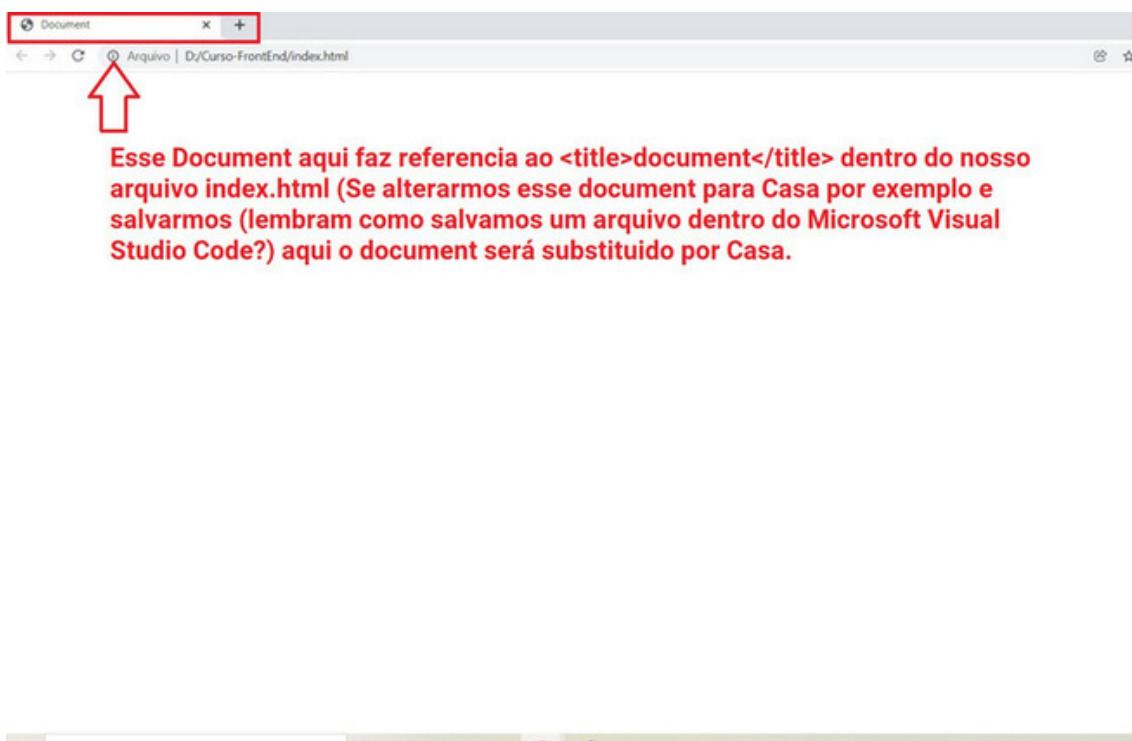
/igor-rebolla

# Explicação de Cada Elemento(Tag) da Estrutura Básica do Documento HTML

## `<title>Document</title>`

Outro elemento (Tag) localizado dentro do Head é o `<title>Document</title>`.

O elemento `<title>` serve para definir o título de sua página. Portanto, é aqui que se define o título que é mostrado na guia do navegador (Google Chrome). Conforme exemplo abaixo:



Esse Document aqui faz referencia ao `<title>document</title>` dentro do nosso arquivo index.html (Se alterarmos esse document para Casa por exemplo e salvarmos (lembrem como salvamos um arquivo dentro do Microsoft Visual Studio Code?) aqui o document será substituido por Casa.



/igor-rebolla

# Explicação de Cada Elemento(Tag) da Estrutura Básica do Documento HTML

## **</head>**

Essa Tag com a / indica o fechamento da tag head, o que significa que tudo que está dentro da tag inicial `<head>` e tag final `</head>` será executado mas não mostrado no navegador, conforme citamos na página anterior.

## **<body>**

Dentro dessa Tag é que vão ficar todos os elementos (Exemplo: parágrafos) que serão renderizados e mostrados pelo navegador, ou seja, todo o corpo(estrutura) da nossa página HTML.

## **</body>**

Essa Tag com a / indica o fechamento da tag `<body>`. O que significa que tudo que está dentro da tag inicial `<body>` e tag final `</body>` será executado e mostrado no navegador.

## **</html>**

Essa Tag com a / indica o fechamento que envolve todo o conteúdo da página HTML.



# Sugestões de prática para essa aula:

- 1.Abrir o Microsoft Visual Studio Code, depois a pasta Curso-FrontEnd e o arquivo index.html (Feito isso digitar linha por linha do que foi mostrado na Estrutura Básica do Documento HTML (São 9 linhas digitadas no total).
- 2.Salvar o que foi digitado no arquivo index.html
- 3.Tentar alterar o **title** do arquivo index.html (Passando de Document para Casa) salvar o arquivo, procurar dentro do Windows o local onde a Pasta Curso-FrontEnd foi criada, clicar 2x no arquivo index.html o navegador irá abrir automaticamente e onde estava document (Será mostrado Casa)

Notem que eu repito algumas coisas e isso pode parecer "cansativo" no começo e é feito de "propósito" mesmo.

O meu objetivo aqui é fazer com que vocês "Comecem a pensar em HTML" que quando vocês forem fazer uma receita de brigadeiro, saia desse jeito:

- Leite condensado OK
- Chocolate OK
- Coloque os ingredientes dentro da Tag <body></body> e leve ao navegador onde o brigadeiro será renderizado e mostrado na tela (Lembra o que a Tag <body></body> faz??)
- Nãoooooo espera e leve ao fogo e mexa até "ficar pronto". rs



/igor-rebolla

# Conceito Adicional e Exemplo

Um documento HTML é composto por diversos elementos (Tags) diferentes. Elas servem para marcar a estrutura (conteúdo) do HTML.

Se digitarmos apenas uma Tag no nosso Arquivo index.html dentro do Microsoft Visual Studio Code, ela sozinha não terá muito sentido. Por isso vamos nessa aula entender os conceitos de 10 Tags e no final desse PDF, vamos estruturar 4 dessas Tags em um exemplo e exibi-lo no navegador.

Tomando como base o exemplo da nossa CASA, ao construirmos uma parede (Se colocarmos apenas um bloco não fará muito sentido, agora se colocarmos o primeiro bloco, verificarmos o nível desse primeiro bloco, acrescentarmos a massa, colocarmos o segundo bloco, verificarmos o nível desse segundo bloco... e assim até a parede ser finalizada (é desse jeito que um documento HTML também se comporta).

A estrutura do HTML funciona como o exemplo acima dos blocos para construção da parede, e dentro do HTML isso é chamado de Estrutura Semântica.



# Semântica no HTML?

Vamos falar sobre semântica no HTML, Tags bem estruturadas dentro do HTML tem um conceito muito importante, o qual é chamado de Elementos Semânticos. A boa utilização desses elementos semânticos, vai influenciar como o nosso site será colocado nos mecanismos de pesquisa, dos sites de busca(Google).

A Semântica é importantíssima também para os leitores de tela no quesito de acessibilidade (quando o leitor faz a "leitura" da tela), se usarmos os elementos semânticos da forma mais estruturada possível para marcamos o nosso conteúdo, o leitor vai conseguir interpretar com mais facilidade o que está sendo exibido na tela.

Agora que vimos um conceito adicional, mais um exemplo fazendo referência entre a estrutura da nossa casa e a estrutura do HTML e o conceito de Semântica.

Veremos na próxima página - Como a Estrutura "Anatomia" dessas(Tags) são compostas.



# Como a "Anatomia" dessas(Tags) são formadas

Aqui abaixo vamos utilizar a Tag <p> como exemplo:

```
<p>Meu primeiro parágrafo</p>
```

1                  2                  3

1 - Tag(Abertura), composta por:

- < sinal de menor
- Nome da Tag, aqui o "p" representa um parágrafo
- > sinal de maior

2 - Conteúdo que será mostrado no navegador (Google Chrome)

- Meu primeiro parágrafo

3 - Tag(Fechamento), composta pela mesma estrutura da Tag de Abertura com o acréscimo da barra.

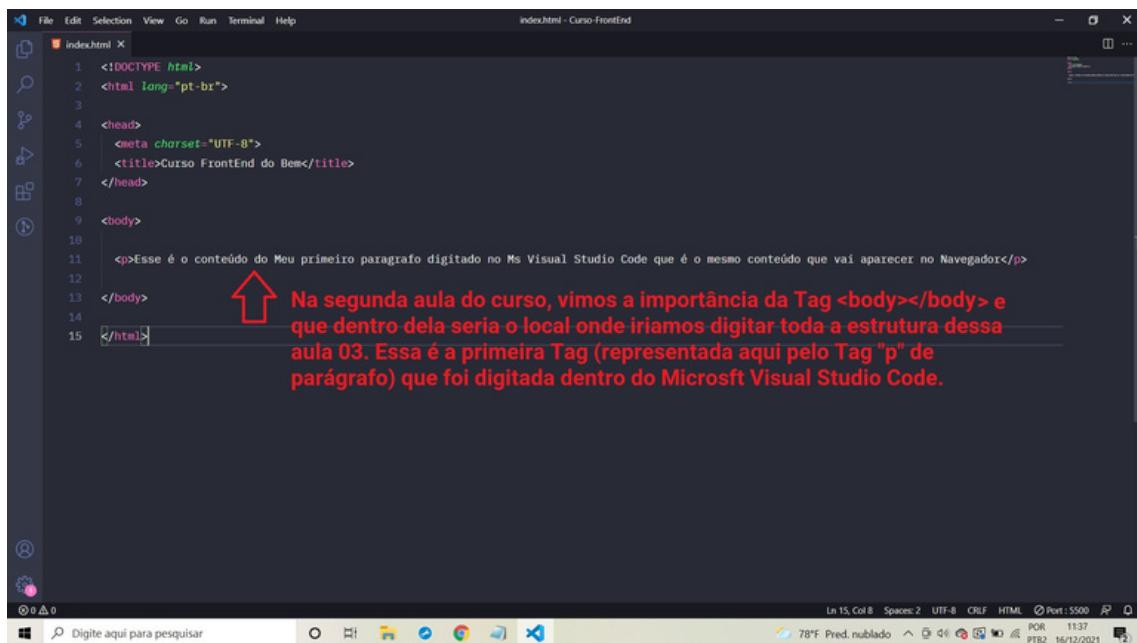
- < sinal de menor
- / que indica o fechamento da tag "p"
- Nome da Tag, aqui o "p" representa um parágrafo
- > sinal de maior



/igor-rebolla

# Como a "Anatomia" dessas(Tags) são formadas

Com base no que foi mostrado na página anterior, vamos digitar outro conteúdo dentro da Tag `<p></p>` para melhor entendimento. Esse conteúdo digitado dentro da Tag `<body></body>` no Microsoft Visual Studio Code será renderizado e mostrado da mesma forma no navegador.



```
File Edit Selection View Go Run Terminal Help
index.html x
1 <!DOCTYPE html>
2 <html lang="pt-br">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>Curso FrontEnd do Bem</title>
7 </head>
8
9 <body>
10
11   <p>Esse é o conteúdo do Meu primeiro parágrafo digitado no Ms Visual Studio Code que é o mesmo conteúdo que vai aparecer no Navegador</p>
12
13 </body>
14
15 </html>
```

Na segunda aula do curso, vimos a importância da Tag `<body></body>` e que dentro dela seria o local onde iríamos digitar toda a estrutura dessa aula 03. Essa é a primeira Tag (representada aqui pelo Tag "p" de parágrafo) que foi digitada dentro do Microsoft Visual Studio Code.



/igor-rebolla

# Como a "Anatomia" dessas(Tags) são formadas

Depois que toda a estrutura "Anatomia" Tag foi digitada, podemos ver no navegador o que foi digitado, para isso temos que salvar o nosso arquivo index.html (Lembram como salvamos o arquivo dentro do Microsoft Visual Studio Code?). Arquivo salvo, vamos procurar a pasta Curso-FrontEnd e clicar 2x no arquivo index.html (o conteúdo da Tag "p") será exibido no Navegador conforme imagem abaixo:



Agora que entendemos a estrutura "Anatomia" de como é composta uma Tag, vamos ver o conceito e logo em seguida um exemplo (para melhor entendimento) de outras 10 Tags Básicas do HTML.



# Como a "Anatomia" dessas(Tags) são formadas

## Heading Tags (h1, h2, h3, h4, h5 e h6)

As Heading Tags são representadas por (h1, h2, h3, h4, h5 h6).

Os sites tem um título principal <h1> e as demais que vão do <h2> até o <h6> representam os subtítulos.

<h1> é a abreviação do inglês para Header 1 (Cabeçalho 1)  
<h2> é a abreviação do inglês para Header 2 (Cabeçalho 2)  
<h3> é a abreviação do inglês para Header 3 (Cabeçalho 3)  
<h4> é a abreviação do inglês para Header 4 (Cabeçalho 4)  
<h5> é a abreviação do inglês para Header 5 (Cabeçalho 5)  
<h6> é a abreviação do inglês para Header 6 (Cabeçalho 6)

Vamos ver um exemplo abaixo de como essas 06 tags podem ser visualizadas tomando como base a localização da nossa casa:

```
<h1>País</h1>
<h2>Estado</h2>
<h3>Cidade</h3>
<h4>Bairro</h4>
<h5>Nome da Rua</h5>
<h6>Número</h6>
```



# Como a "Anatomia" dessas(Tags) são formadas

## Heading Tags (h1, h2, h3, h4, h5 e h6)

Com base no que foi mostrado na página anterior, vamos abrir o Microsoft Visual Studio Code e logo em seguida o arquivo index.html e digitaremos a estrutura das Tags `<h1>conteúdo</h1>`, `<h2>conteúdo</h2>`, `<h3>conteúdo</h3>`, `<h4>conteúdo</h4>`, `<h5>conteúdo</h5>` e `<h6>conteúdo</h6>` dentro da Tag `<body></body>`, conforme imagem abaixo:

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <meta charset="UTF-8">
6      <title>Curso FrontEnd do Bem</title>
7  </head>
8
9  <body>
10     <h1>País</h1>
11     <h2>Estado</h2>
12     <h3>Cidade</h3>
13     <h4>Bairro</h4>
14     <h5>Nome da Rua</h5>
15     <h6>Número</h6>
16
17
18 </body>
19
20 </html>
```



/igor-rebolla

# Como a "Anatomia" dessas(Tags) são formadas.

## Tags de Lista <ol>, <ul> e <li>

Dentro do nosso arquivo index.html, podemos criar 2 tipos de listas:

<ol> - Ordered List (Lista ordenada), o qual os itens são organizados por números.

<ul> - Unordered List(Lista não-ordenada), o qual os itens são organizados por bulleted list (Lista de marcadores – aquelas bolinhas)

<li> - List Item (Item da Lista), representa cada item da lista e está presente tanto na <ol> quanto na <ul>.

Nas próximas páginas para melhor entendimento veremos 2 exemplos distintos tanto da <ol> quanto da <ul> (notem que em ambas a Tag <li> se faz presente ou seja "é brother das 2".)



# Como a "Anatomia" dessas(Tags) são formadas

## Tags de Lista <ol>, <ul> e <li>

Exemplo da estrutura de códigos Lista Ordenada <ol></ol>.

Tomando como base uma lista com 3 itens (materiais de construção) para o exemplo da nossa casa:

```
<ol>
<li>10 Sacos de Cimento 50kg</li>
<li>35 Blocos</li>
<li>20 Sacos de Areia</li>
</ol>
```

Com base no que foi mostrado acima, vamos abrir o Microsoft Visual Studio Code e logo em seguida o arquivo index.html e digitaremos a estrutura das Tags <ol></ol>, <li>conteúdo</li>, <li>conteúdo</li> e <li>conteúdo</li>, dessa forma dentro da tag <body></body>, conforme imagem exibida na próxima página:



# Como a "Anatomia" dessas(Tags) são formadas

## Tags de Lista <ol>, <ul> e <li>

```
index.html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>Curso FrontEnd do Bem</title>
7  </head>
8
9  <body>
10
11    <ol>
12      <li>10 Sacos de Cimento 50kg</li>
13      <li>35 Blocos</li>
14      <li>20 Sacos de Areia</li>
15    </ol>
16
17  </body>
18
19 </html>
```

Essa é a nossa lista ordenada representada pela Tag <ol>. E cada tag <li> representa cada item da lista que está dentro da tag <ol>

Depois que toda a estrutura "Anatomia" da Tag foi digitada, podemos ver no navegador o que foi digitado, para isso temos que salvar o nosso arquivo index.html (Lembram como salvamos o arquivo dentro do Microsoft Visual Studio Code?), arquivo salvo, vamos procurar a pasta Curso-FrontEnd e clicar 2x no arquivo index.html (o conteúdo de cada Tag <li>) será exibido no Navegador conforme imagem da próxima página:



# Como a "Anatomia" dessas(Tags) são formadas

## Tags de Lista <ol>, <ul> e <li>



Exemplo da estrutura de códigos Lista Não-Ordenada <ul>.

Tomando como base uma lista com 3 itens (materiais de construção) para o exemplo da nossa casa:

```
<ul>
<li>10 Sacos de Cimento 50kg</li>
<li>35 Blocos</li>
<li>20 Sacos de Areia</li>
</ul>
```

Com base no que foi mostrado acima, vamos abrir o Microsoft Visual Studio Code e logo em seguida o arquivo index.html e digitaremos a estrutura das Tags <ul></ul>, <li>conteúdo</li>, <li>conteúdo</li> e <li>conteúdo</li> dentro da Tag <body></body>, conforme imagem da próxima página:



# Como a "Anatomia" dessas(Tags) são formadas

## Tags de Lista <ol>, <ul> e <li>

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Curso FrontEnd do Bem</title>
</head>
<body>
    <ul>
        <li>10 Sacos de Cimento 50kg</li>
        <li>35 Blocos</li>
        <li>20 Sacos de Areia</li>
    </ul>
</body>
</html>
```

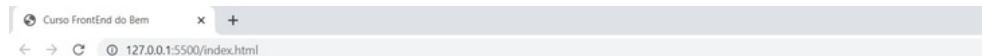
Essa é a nossa lista não-ordenada representada pela Tag <ul>. E cada tag <li> representa cada item da lista que está dentro da tag <ul>.

Depois que toda a estrutura "Anatomia" da Tag foi digitada, podemos ver no navegador o que foi digitado, para isso temos que salvar o nosso arquivo index.html (Lembram como salvamos o arquivo dentro do Microsoft Visual Studio Code?), arquivo salvo, vamos procurar a pasta Curso-FrontEnd e clicar 2x no arquivo index.html (o conteúdo de cada Tag <li>) será exibido no Navegador conforme imagem da próxima página:



# Como a "Anatomia" dessas(Tags) são formadas

Tags de Lista <ol>, <ul> e <li>



# Resumo das Tags vistas até o momento

**As 10 Tags abaixo foram vistas até o momento**

- <p></p>
- <h1></h1>
- <h2></h2>
- <h3></h3>
- <h4></h4>
- <h5></h5>
- <h6></h6>
- <ul></ul>
- <ol></ol>
- <li></li>

Bora juntar 4 dessas Tags e começar a estruturar algo um pouquinho maior. Eu ouvi um sim, ai do outro lado?

Simmmmmmmmmmmmmmmmmmmmmmmmmmmmm!!!

Então, partiu:



# Exemplo usando 4 Tags no mesmo documento HTML

**As 4 Tags que vamos usar são:**

<h1></h1> - O título principal da nossa lista  
<p></p> - Um parágrafo com uma breve descrição da nossa lista  
<ol></ol> - Vamos criar uma lista ordenada  
<li></li> - Vamos criar 4 itens da nossa lista, e cada item será representado por uma <li>

Só vamos relembrar antes, quais os passos dados para chegarmos até o momento de digitar todo o nosso código dentro da Tag <body></body>:

- 1 – Abrir o Microsoft Visual Studio Code;
- 2 – Abrir a pasta Curso-FrontEnd;
- 3 – Abrir o arquivo index.html (dentro da pasta Curso-FrontEnd);
- 4 – Caso a estrutura básica do lado direito da tela (Aquela que começa com DOCTYPE e tem 9 linhas de código) não tenha sido digitada ainda (peço a gentileza para consultar a Aula 02 desse curso)



# Exemplo usando 4 Tags no mesmo documento HTML

E bora digitar o nosso código abaixo dentro da Tag `<body></body>`:

```
<h1>Lista de Materiais para o Alicerce da Casa</h1>

<p>A lista abaixo foi elaborada para iniciarmos a construção
do Alicerce da nossa casa</p>

<ol>
  <li>050 Sacos de Cimento 50kg</li>
  <li>670 Blocos</li>
  <li>003 Metros de Areia</li>
  <li>002 Metros de Pedra</li>
</ol>
```

O código acima ficará assim dentro do Microsoft Visual Studio Code, conforme imagem da próxima página:



# Exemplo usando 4 Tags no mesmo documento HTML

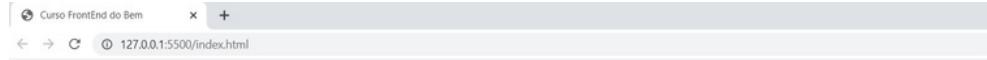


```
index.html
1 <!DOCTYPE html>
2 <html lang="pt-br">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>Curso FrontEnd do Bem</title>
7 </head>
8
9 <body>
10
11   <h1>Lista de Materiais para o Alicerce da Casa</h1>
12
13   <p>A lista abaixo foi elaborada para iniciarmos a construção do Alicerce da nossa casa</p>
14
15   <ol>
16     <li>050 Sacos de Cimento 50kg</li>
17     <li>670 Blocos</li>
18     <li>003 Metros de Areia</li>
19     <li>002 Metros de Pedra</li>
20   </ol>
21
22 </body>
23
24 </html>
```

O que foi digitado dentro da Tag `<body></body>` na imagem acima será renderizado e mostrado no navegador. Só relembrando para que isso aconteça; temos que salvar o nosso arquivo `index.html` (Lembram como salvamos o arquivo?) e clicar 2x nele dentro da pasta `Curso-FrontEnd`. Arquivo `index.html` devidamente salvo e clicado 2x, a nossa lista com um título `<h1>` e um parágrafo `<p>` será mostrado no navegador conforme imagem da próxima página:



# Exemplo usando 4 Tags no mesmo documento HTML



## Lista de Materiais para o Alicerce da Casa ➔ <h1></h1>

A lista abaixo foi elaborada para iniciarmos a construção do Alicerce da nossa casa ➔ <p></p>

- 1. 050 Sacos de Cimento 50kg
- 2. 670 Blocos
- 3. 003 Metros de Areia
- 4. 002 Metros de Pedra

➔ <ol>  
    <li></li>  
    <li></li>  
    <li></li>  
    <li></li>  
</ol>



/igor-rebolla

# Sugestões de prática para essa aula:

1. Abrir o Microsoft Visual Studio Code, depois a pasta Curso-FrontEnd e o arquivo index.html (Feito isso digitar cada uma das 10 Tags mostradas nesse aula)
2. Salvar o que foi digitado no arquivo index.html
3. Tentar fazer as seguintes alterações no último exemplo que foi visto nessa aula:
  - Alterar o Titulo Principal `<h1>` para `<h2>`
  - Alterar a lista ordenada `<ol>` para lista não-ordenada `<ul>`
  - Trocar os conteúdos de todas as Tags (Criando o conteúdo da sua própria lista, pode alterar a lista de materiais de construção para itens da sua preferência)

Notem que assim como nas aula (01 e 02) aqui também eu repito algumas coisas e isso pode parecer "cansativo" no começo e é feito de "propósito" mesmo para melhor entendimento dos conceitos.



# O que são Atributos em uma TAG HTML

Vimos na Aula 03 desse curso (Como a anatomia/estrutura) das Tags abaixo são formadas:

<p>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <ul>, <ol> e <li>

Exemplo: <p>Conteúdo que será visto no navegador</p>

Nessa aula vamos entender como funciona uma Tag com um item adicional, o qual em HTML chamamos de ATRIBUTOS. E o que são esses Atributos:

Um atributo envolve uma informação adicional que nós informamos para a nossa TAG, para que ela possa cumprir seu papel ou função de maneira eficaz.

Para entendermos melhor, pense neste exemplo:

Vamos imaginar que o encanamento da nossa casa está com um vazamento e precisamos chamar um Encanador, pois o vazamento está aumentando .

Qual é o objetivo do Encanador nesse caso? É te auxiliar a ficar livre do vazamento..



# O que são Atributos em uma TAG HTML

Mas, para isso, o que temos que informar pra ele?

É necessário dizer onde o vazamento está acontecendo...

Apenas com essa informação o Encanador conseguirá saber qual ação irá tomar nesse caso do vazamento.

Ou seja, somente com essa informação que ele conseguirá cumprir seu papel e sanar o problema do vazamento. É exatamente o que ocorre em um atributo em uma TAG HTML, se não informamos o Atributo certo para a TAG ela ficará "sem saber" o que fazer.

Por exemplo: qual é a função de uma TAG de Imagem? É exibir a Imagem.

Mas, para isso, o que ela precisa saber? O caminho da imagem que será exibido no navegador, e para isso usamos o atributo src.

Aproveitando o "Gancho" do parágrafo acima, na próxima página vamos entender melhor como funciona a TAG de imagem, sua estrutura (anatomia) e o local onde colocamos o Atributo src.



# Tag - Imagem <img>

Em HTML usamos a TAG <img> para informar ao nosso arquivo index.html que ali estamos indicando uma imagem que será exibida no navegador.

A anatomia da TAG <img> é composta da seguinte estrutura abaixo:

```
  
1           2           3
```

1 - Tag(Abertura), composta por:

- < sinal de menor
- Nome da Tag, aqui o "img" representa uma imagem
- src que é o atributo que vai indicar o local(caminho) da imagem

2 - Imagem que será mostrada no navegador (Google Chrome)

- Aqui o nome foi definido como: alicerce.jpg (onde .jpg é a extensão que foi escolhida para salvarmos essa imagem)

3 - alt = Este atributo define um texto alternativo que descreve a imagem "alicerce.jpg" caso a mesma não seja mostrada pelo navegador. Caso isso aconteça o texto "Meu Alicerce" será exibido no navegador.  
> sinal de maior que indica o fechamento da tag "img"

Obs.: Notem que diferente da Tag <p> que tem o seu fechamento definido por </p> a qual vimos na Aula 03, a Tag <img> não tem o seu fechamento indicado por </img> e sim por um sinal de maior >. É importante prestarmos atenção nesses "pequenos detalhes".



# Tag - Imagem <img>

E bora ver um exemplo dentro do Ms Visual Studio Code da Tag <img> com os atributos src e alt.

Ah, antes disso temos que procurar uma imagem referente ao Alicerce da Casa e salva-la dentro da pasta Curso-FrontEnd como: alicerce.jpg

Depois que a imagem (alicerce.jpg) está salva dentro da pasta Curso-FrontEnd conforme item 1 vamos digitar o nosso código  dentro da Tag <body> conforme item 2 indicado na imagem abaixo:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Curso FrontEnd do Bem</title>
</head>
<body>

</body>
</html>
```

1 - Nosso arquivo  
alicerce.jpg

Que foi salvo dentro  
da pasta Curso-Front  
End.

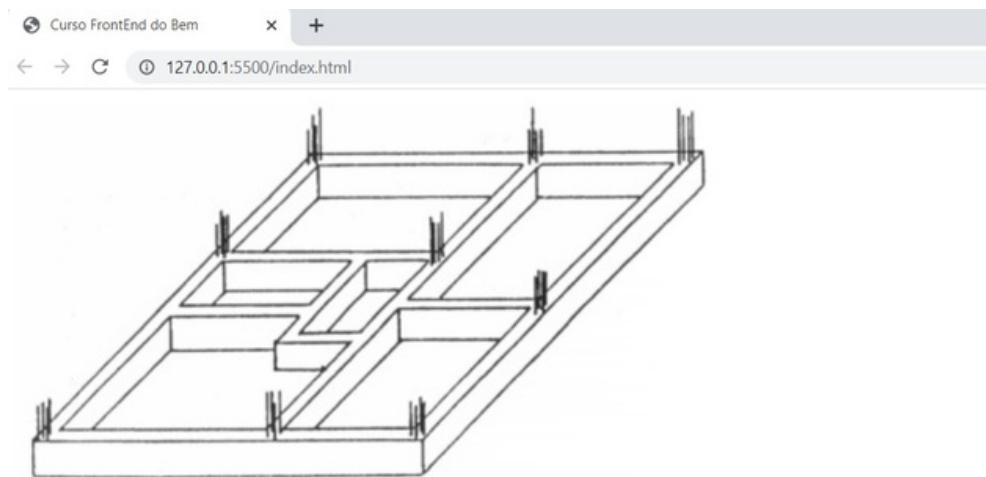
2 - Esse é o código referente a Tag <img> com o atributo SRC  
responsável por mostrar o caminho da imagem e o atributo ALT  
responsável por mostrar um texto aqui representado por Meu  
Alicerce, caso a imagem alicerce.jpg não seja mostrada no  
navegador.



/igor-rebolla

# Tag - Imagem <img>

Com o código da página anterior digitado e o arquivo index.html devidamente salvo (Lembram como salvamos esse arquivo?). Vamos ver o resultado da Tag <img> com os seus atributos, exibida no Navegador, conforme figura abaixo:



Agora que vimos o conceito + anatomia(estrutura) + código digitado no Ms Visual Studio Code + exemplo desse código mostrado no Navegador da Tag <img> com seus atributos SRC e ALT.

Na próxima página vamos ver o conceito e a estrutura da TAG <a> e seu atributo HREF.



# Tag - link <a>

A Tag <a> faz referência para um link, logo após sua abertura indicada por < temos o atributo HREF onde colocamos o valor do link externo o qual queremos acessar Ex.: <https://www.linkedin.com>.

A anatomia da TAG <a> é composta da seguinte estrutura abaixo:

```
<a href = "https://www.linkedin.com">Linkedin</a>  
1           2   3
```

1 - Tag(Abertura), composta por:

- < sinal de menor
- Nome da Tag, aqui o "a" representa um link
- href que é o atributo onde colocamos o valor do link (Aqui representado por: <https://www.linkedin.com>)

2 - Conteúdo que será mostrado no navegador (Google Chrome)

- Linkedin

3 - Tag(Fechamento), composta pela mesma estrutura da Tag de Abertura com o acréscimo da barra.

- < sinal de menor
- / que indica o fechamento da tag "a"
- Nome da Tag, aqui o "a" representa um link
- > sinal de maior



# Tag - link <a>

E bora ver um exemplo dentro do Ms Visual Studio Code da Tag <a> com o atributo href. Conforme imagem abaixo:



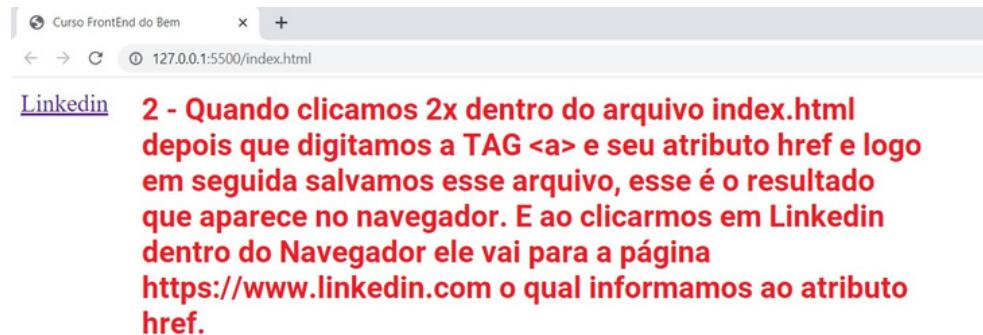
```
EXPLORER OPEN EDITORS CURSO-FRONTEND index.html
index.html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>Curso FrontEnd do Bem</title>
7  </head>
8
9  <body>
10
11    <a href="https://www.linkedin.com">Linkedin</a>
12
13  </body>
14
15 </html>
```

1 - Esse é o código referente a Tag <a> que faz referência para um link e logo em seguida acrescentamos o atributo HREF que é o responsável por indicar o valor "https://www.linkedin.com" o qual queremos acessar quando clicarmos em Linkedin no navegador.

Com o código da página anterior digitado e o arquivo index.html devidamente salvo (Lembram como salvamos esse arquivo?). Vamos ver o resultado da Tag <a> com o seu atributo, exibida no Navegador, conforme figura da próxima página:



# Tag - link <a>



Notem na imagem acima que a palavra Linkedin está sublinhada (Esse sublinhado indica que um Link da tag <a> foi referenciado aqui. Ao clicarmos na palavras Linkedin (o navegador abrirá a página do Linkedin). Conforme imagem abaixo:



# - Exemplo usando 7 Tags no mesmo documento HTML

**As 7 Tags que vamos usar são:**

<h1></h1> - O título principal da nossa lista  
<h2></h2> - O subtítulo da nossa lista  
<p></p> - Um parágrafo com uma breve descrição da nossa lista  
<a></a> - Um link para acessarmos um link externo (href)  
<img> - Mostrar uma imagem (src) e um texto alternativo (alt) caso essa imagem não seja exibida no navegador  
<ul></ul> - Vamos criar uma lista não-ordenada  
<li></li> - Vamos criar 7 itens da nossa lista, e cada item será representado por uma <li>

Só vamos relembrar antes, quais os passos dados para chegarmos até o momento de digitar todo o nosso código dentro da Tag <body></body>:

- 1 - Abrir o Microsoft Visual Studio Code;
- 2 - Abrir a pasta Curso-FrontEnd;
- 3 - Abrir o arquivo index.html (dentro da pasta Curso-FrontEnd);
- 4 - Salvar a imagem como html.jpg
- 5 - Caso a estrutura básica do lado direito da tela (Aquela que começa com DOCTYPE e tem 9 linhas de código) não tenha sido digitada ainda (peço a gentileza para consultar a Aula 02 desse curso)

E bora digitar o código da próxima página dentro da Tag <body></body>:



# - Exemplo usando 7 Tags no mesmo documento HTML

```
<h1>Lista de Tags usadas até essa Aula 04</h1>
<h2>Usamos o Microsoft Visual Studio Code para digitarmos nossas Tags
</h2>



<a href="https://www.google.com.br">Google</a>

<p>A lista não-ordenada abaixo mostra quais as TAGS que usamos até aqui para construirmos o alicerce da nossa casa ou seja o primeiro pilar (HTML) dos 3 pilares (HTML, CSS e JavaScript) que veremos nesse curso</p>

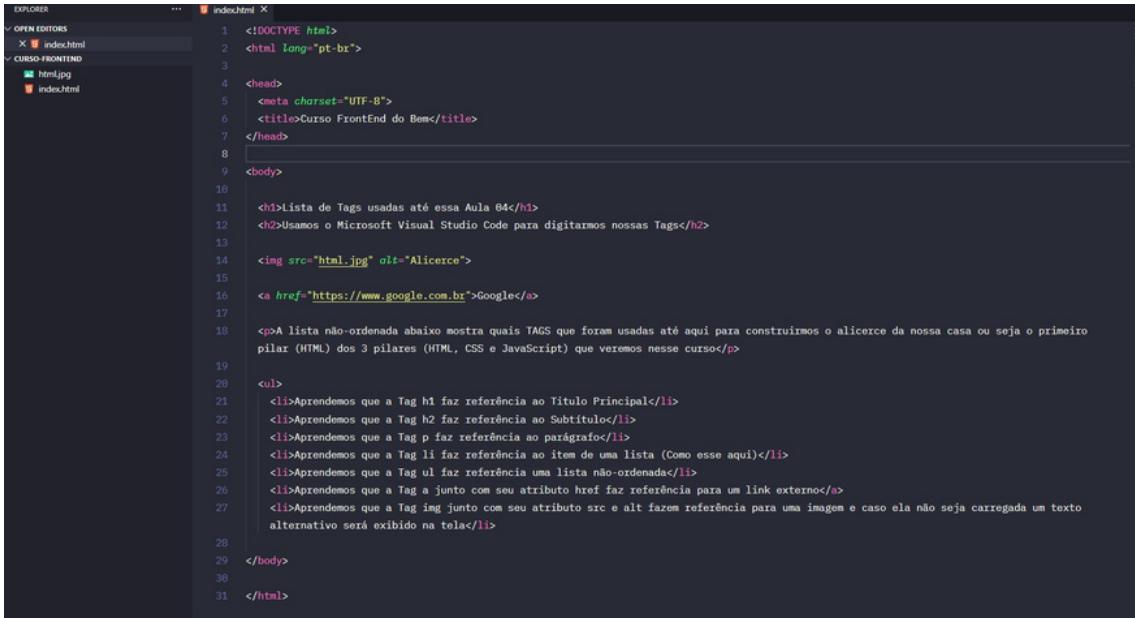
<ul>
<li>Aprendemos que a Tag h1 faz referência ao Titulo Principal</li>
<li>Aprendemos que a Tag h2 faz referência ao Subtítulo</li>
<li>Aprendemos que a Tag p faz referência ao parágrafo</li>
<li>Aprendemos que a Tag li faz referência ao item de uma lista (Como esse aqui)</li>
<li>Aprendemos que a Tag ul faz referência uma lista não-ordenada</ul>
<li>Aprendemos que a Tag a junto com seu atributo href faz referência para um link externo</li>
<li>Aprendemos que a Tag img junto com seu atributo src e alt fazem referência para uma imagem e caso ela não seja carregada um texto alternativo será exibido na tela</li>
```



/igor-rebolla

# - Exemplo usando 7 Tags no mesmo documento HTML

O código da página anterior ficará assim dentro do Microsoft Visual Studio Code, conforme imagem abaixo:



```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Curso FrontEnd do Bem</title>
</head>
<body>
    <h1>Lista de Tags usadas até essa Aula 04</h1>
    <h2>Usamos o Microsoft Visual Studio Code para digitarmos nossas Tags</h2>
    
    <a href="https://www.google.com.br">Google</a>
    <p>A lista não-ordenada abaixo mostra quais TAGS que foram usadas até aqui para construirmos o alicerce da nossa casa ou seja o primeiro pilar (HTML) dos 3 pilares (HTML, CSS e JavaScript) que veremos nesse curso</p>
    <ul>
        <li>Aprendemos que a Tag h1 faz referência ao Título Principal</li>
        <li>Aprendemos que a Tag h2 faz referência ao Subtítulo</li>
        <li>Aprendemos que a Tag p faz referência ao parágrafo</li>
        <li>Aprendemos que a Tag li faz referência ao item de uma lista (Como esse aqui)</li>
        <li>Aprendemos que a Tag ul faz referência uma lista não-ordenada</li>
        <li>Aprendemos que a Tag a junto com seu atributo href faz referência para um link externo</a>
        <li>Aprendemos que a Tag img junto com seu atributo src e alt fazem referência para uma imagem e caso ela não seja carregada um texto alternativo será exibido na tela</li>
    </ul>
</body>
</html>
```

O que foi digitado dentro da Tag `<body></body>` na imagem acima será renderizado e mostrado no navegador. Só relembrando para que isso aconteça; temos que salvar o nosso arquivo `index.html` (Lembram como salvamos o arquivo?) e clicar 2x nele dentro da pasta `Curso-FrontEnd`. Arquivo `index.html` devidamente salvo e clicado 2x, o nosso exemplo será mostrado no navegador conforme imagem da próxima página:



# - Exemplo usando 7 Tags no mesmo documento HTML

Lista de Tags usadas até essa Aula 04      h1

Usamos o Microsoft Visual Studio Code para digitarmos nossas Tags      h2

**HTML**

     a href

Google

A lista não-ordenada abaixo mostra quais TAGS que foram usadas até aqui para construirmos o alicerce da nossa casa ou seja o primeiro pilar (HTML) dos 3 pilares (HTML, CSS e JavaScript) que veremos nesse curso      p

- Aprendemos que a Tag h1 faz referência ao Titulo Principal
- Aprendemos que a Tag h2 faz referência ao Subtítulo
- Aprendemos que a Tag p faz referência ao parágrafo
- Aprendemos que a Tag li faz referência ao item de uma lista (Como esse aqui)
- Aprendemos que a Tag ul faz referência uma lista não-ordenada
- Aprendemos que a Tag a junto com seu atributo href faz referência para um link externo
- Aprendemos que a Tag img junto com seu atributo src e alt fazem referência para uma imagem e caso ela não seja carregada um texto alternativo será exibido na tela ...



/igor-rebolla

# **Sugestões de prática para essa aula:**

1. Abrir o Microsoft Visual Studio Code, depois a pasta Curso-FrontEnd e o arquivo index.html (Feito isso digitar a tag `<a>` com o seu respectivo atributo e a tag `<img>` com seus respectivos atributos)
2. Procurar uma imagem na internet e coloca-la no atributo `src`
3. Salvar o que foi digitado no arquivo index.html ( e ver o resultado no navegador)
4. Tentar fazer um exemplo usando 7 Tags html conforme último exemplo mostrado nessa aula (E ver o resultado no navegador).



**/igor-rebolla**

# Tag <div>

A Tag <div> é usada para criar uma divisão no nosso documento HTML(estrutura), então sempre que queremos reservar algum espaço (vamos usar a Tag <div>)

A Tag <div> também é utilizada como um Container para outros elementos HTML. Sempre que queremos "agrupar" e colocar algum estilo CSS (Assunto do Nosso Segundo Pilar desse curso) é muito comum o uso da Tag <div>.

A Tag <div>, assim como a Tag <p> por exemplo tem a sua abertura e o seu fechamento (e o conteúdo vai entre essas 2 tags), representado pela anatomia (estrutura) abaixo:

```
<div>Conteúdo Vai Aqui</div>
```

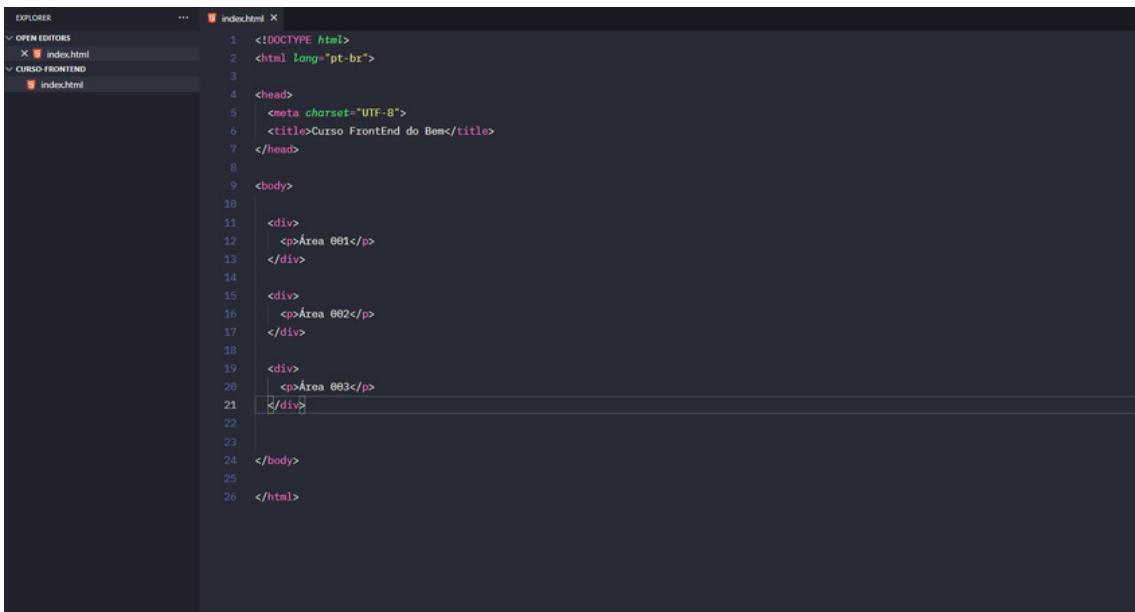
Como é muito utilizada para agrupar elementos, acaba sendo um facilitador para organizar informações dentro de um layout (Aqui vamos usar o exemplo da nossa Casa), mas pode ser replicado para o layout de um site (Sem problemas).

Vamos usar 3 divisões <div> para poder separar os nossos parágrafos <p>.



# Tag <div>

E bora ver um exemplo dentro do Ms Visual Studio Code da Tag <div> com essas 3 divisões. Conforme imagem abaixo:



The screenshot shows the Microsoft Visual Studio Code interface. On the left, the Explorer sidebar shows 'OPEN EDITORS' with 'index.html' and 'CURSO-FRONTEND' with 'index.html'. The main area displays the code for 'index.html':

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Curso FrontEnd do Bem</title>
</head>
<body>
<div>
<p>Área 001</p>
</div>
<div>
<p>Área 002</p>
</div>
<div>
<p>Área 003</p>
</div>
</body>
</html>
```

Com o código da página anterior digitado e o arquivo index.html devidamente salvo (Lembram como salvamos esse arquivo?). Vamos ver o resultado da Tag <div> exibida no Navegador, conforme figura da próxima página:



# Tag <div>



Notem que na figura acima foram exibidos 3 parágrafos (cada um representado por sua respectiva Tag <p>) e onde a Tag <div> entra nessa história:

Lembram que falamos que para uma Tag ser direcionada de forma mais correta ela precisaria que atributos fossem acrescentados nela, pois bem aqui os atributos são:

- id (Identificador)
- class (classe)

Dessa forma é possível formatar e manipular os elementos, inclusive a própria <div>, através do CSS de uma forma organizada. Geralmente é acompanhada dos atributos de ID e/ou Classe para poder facilitar essa organização e formatação.

Partiu ver o que são id e class (nas próximas páginas).



# O que é um id ?

Um id é uma configuração única, ou seja só podemos usar um id em apenas um único elemento.

**Leiam apenas 1x o parágrafo abaixo (Ele se autodestruirá em 3..2..1).**

O HTML vai permitir que você use mais de um elemento com o mesmo id, embora não seja esperado que venha outros elementos com o mesmo id, mas ele também vai funcionar se tivermos mais de um id. (apenas para conhecimento)

**Ufa, esse parágrafo representa melhor o que é o id**

Massssssssss a proposta original do id é que ele seja único e que tenhamos apenas um id no nosso documento HTML.

Vamos ver como funciona a estrutura do id:

Dentro do nosso arquivo index.html (entre as tags body)

- 1 Passo é criarmos uma div <div></div>
- 2 Passo é atribuir um id para essa div do passo 1 o qual iremos chama-la de "area-1"

Esse é o resultado final da estrutura:

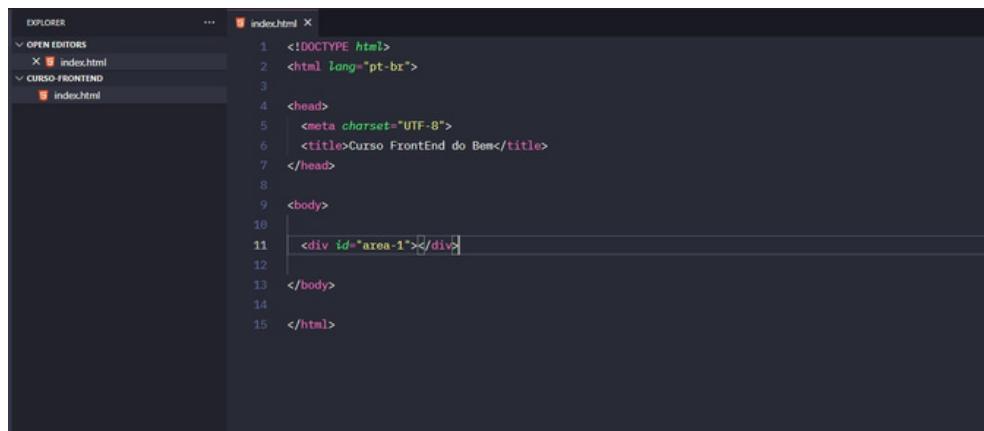
```
<div id="area-1"></div>
```



/igor-rebolla

# O que é um id ?

E bora ver um exemplo dentro do Ms Visual Studio Code do id, conforme imagem abaixo:



```
index.html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>Curso FrontEnd do Bem</title>
7  </head>
8
9  <body>
10
11 <div id="area-1"></div>
12
13 </body>
14
15 </html>
```

Depois que o código acima foi digitado e o documento index.html foi salvo para ver o que foi digitado eu teria que abrir o arquivo index.html no navegador. Se eu fizer isso com esse exemplo, não vai aparecer nada no navegador. Aqui a ideia é entender:

- Como funciona o conceito do id
- Entender a sua estrutura (anatomia) dentro do Ms Visual Studio Code

Apenas para ficar no radar de vocês para a sexta e próxima aula(não se preocupem com isso agora):

O id é definido pela símbolo da hashtag **#** e seguido pelo valor do id (no exemplo acima definimos esse valor como "area-1")

Vai ficar assim: #area-1 (dentro do nosso arquivo CSS)



# O que é uma class (classe) ?

Uma classe é uma configuração que pode ser usada em mais de um elemento.

Vamos ver como funciona a class (classe):

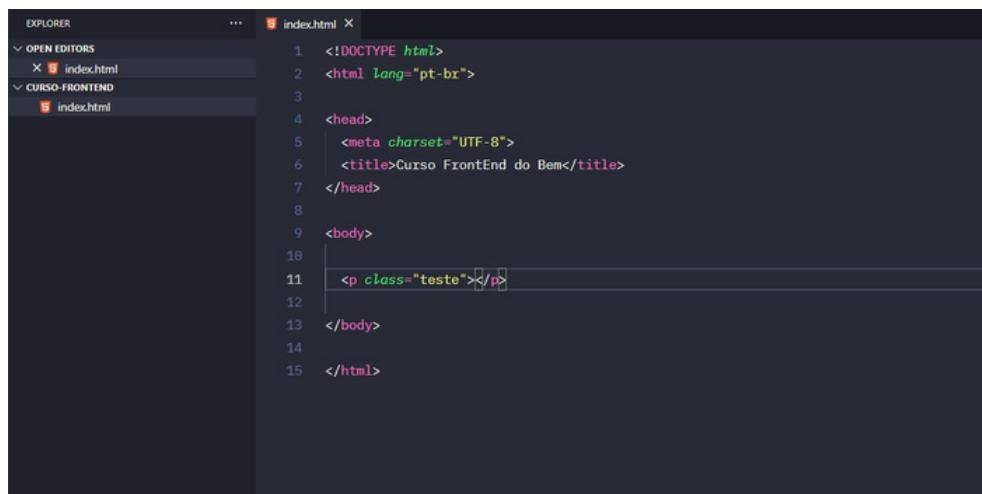
Dentro do nosso arquivo index.html (entre as tags body)

- 1 Passo é criarmos uma Tag <p>
- 2 Passo é atribuir uma class para a Tag <p> do passo 1 o qual iremos chama-la de "teste"

Esse é o resultado final da estrutura:

```
<p class="teste"></p>
```

E bora ver um exemplo dentro do Ms Visual Studio Code da class (classe), conforme imagem abaixo:



```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>Curso FrontEnd do Bem</title>
7  </head>
8
9  <body>
10
11  <p class="teste"></p>
12
13 </body>
14
15 </html>
```



/igor-rebolla

# O que é uma class (classe) ?

Depois que o código da página anterior foi digitado e o documento index.html foi salvo para ver o que foi digitado eu teria que abrir o arquivo index.html no navegador. Se eu fizer isso com esse exemplo, não vai aparecer nada no navegador. Aqui a ideia é entender:

- Como funciona o conceito de class (classe)
- Entender a sua estrutura (anatomia) dentro do Ms Visual Studio Code

Apenas para ficar no radar de vocês para a sexta e próxima aula(não se preocupem com isso agora):

A class (classe) é definida pela símbolo do ponto . e seguido pelo valor da class (no exemplo acima definimos esse valor como "teste")

Vai ficar assim: .teste (dentro do nosso arquivo CSS)



# Dicionário com os Termos usados até essa aula

- Visual Studio Code: Programa da MS onde digitamos os nossos códigos
- ctrl + s: Teclas utilizadas para salvarmos o nosso arquivo index.html
- <p>: Tag que representa um parágrafo
- <h1>: Tag que representa um Titulo Principal
- <h2>: Tag que representa um subtítulo
- <li>: Tag que representa cada item da lista
- <ol>: Tag que representa uma lista ordenada
- <ul>: Tag que representa uma lista não-ordenada
- <a>: Tag que representa um link
- <img>: Tag que representa uma imagem
- <div>: Tag que representa uma divisão (container)
- Atributo: envolve uma informação adicional que nós informamos para a nossa TAG.
- src: é o atributo da Tag <img> que vai indicar o local(caminho) da imagem
- alt: outro atributo da Tag <img> que define um texto alternativo que descreve a imagem caso a mesma não seja mostrada pelo navegador
- href: é o atributo da Tag <a> onde colocamos o valor do link (Aqui representado por: <https://www.linkedin.com>)
- class: é uma configuração que pode ser usada em mais de um elemento
- id: é um configuração única, ou seja só podemos usar um id em apenas um único elemento.



/igor-rebolla

# **Sugestões de prática para essa aula:**

1. Abrir o Microsoft Visual Studio Code, depois a pasta Curso-FrontEnd e o arquivo index.html (Feito isso digitar uma tag <p> com o nome do seu id e outra tag <p> com o nome da sua class (classe))
2. Salvar o que foi digitado no arquivo index.html ( e ver o resultado no navegador (apenas para conhecimento, pois, não mostrará nada, conforme citado nos exemplos dessa mesma aula))
3. Revisar o conteúdo das aulas anteriores (refazendo os exemplos)
4. Dar uma olhada no dicionário que está localizado 2 páginas acima dessa aqui com os termos usados até momento (bom pra revisão)



# O que é CSS ?

Chegamos na primeira aula do Segundo pilar do nosso curso que é o CSS.

Igor, e o que é esse tal de CSS?

O CSS (Cascading Style Sheet) é uma linguagem de estilo e fazendo aquela tradução marota, significa: (Folha de Estilo em Cascatas). O CSS é utilizado para estilizar elementos (as Tags) os quais escrevemos em uma linguagem de marcação (estrutura) como o HTML (o qual vimos nas 5 aulas anteriores desse curso).

Vamos pensar na decoração da nossa casa. Ao utilizarmos o CSS é possível:

- Alterar a cor do fundo (a cor de uma parede por exemplo)
- Alterar o tamanho e a cor da fonte (texto)
- Centralizar/Ajustar os itens nos cômodos da casa
- Dar espaçamento entre parágrafos, títulos, subtítulos...
- Entre outros...

Lembram que no HTML, criamos um arquivo chamado index.html, pois bem aqui no CSS precisamos criar o seu respectivo arquivo.

O qual chamaremos de: style.css

Bora ver na próxima página como vamos criar o arquivo style.css



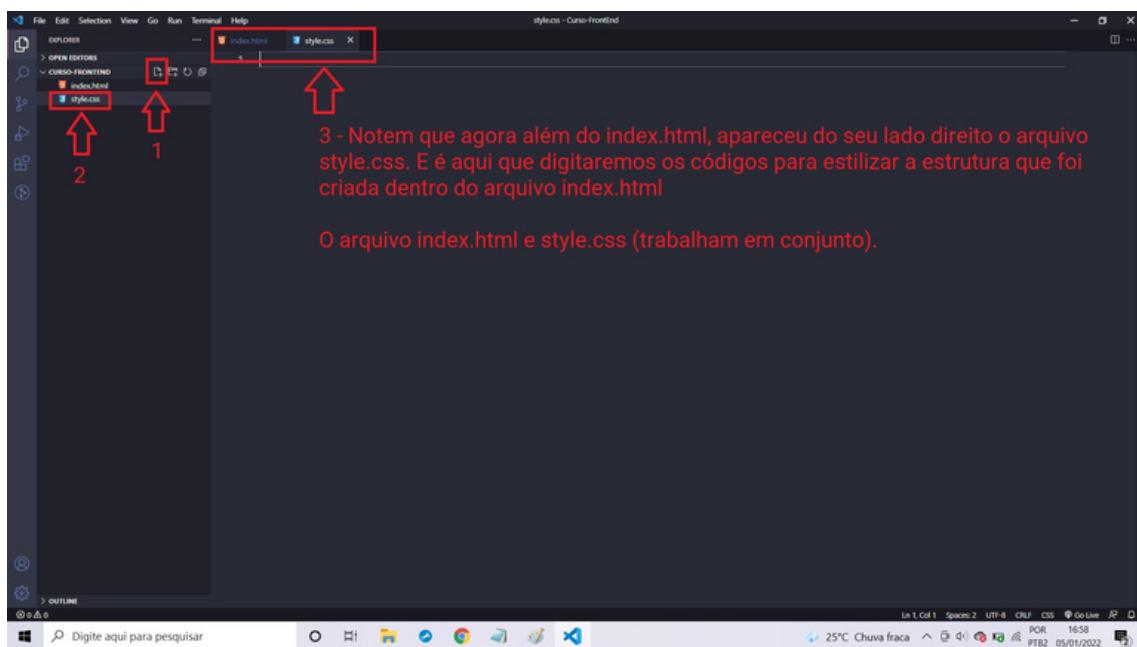
# O que é CSS ?

Vamos abrir o MS Visual Studio Code e aproveitar a estrutura criada na Aula 01 desse curso, que foi a seguinte:  
Pasta (Curso-FrontEnd) e arquivo index.html

Agora nomearemos o segundo arquivo do nosso curso:  
Conforme a indicação do Número 1 na imagem abaixo (vamos clicar) no desenho que representa a criação de um arquivo

Logo abaixo da pasta Curso-FrontEnd o cursor ficará piscando para ser digitado algo, e é aqui que daremos o nome de: style.css (Conforme a indicação do Número 2 na imagem abaixo e logo em seguida pressionaremos o enter 1x para gravar esse nome.

O número 3 da imagem (será o local onde digitaremos os nossos códigos de estilo CSS.



/igor-rebolla

# Como a comunicação é feita entre o HTML e o CSS?

Antes de partirmos para os códigos CSS (Sei que a galera tá na expectativa ai do outro lado). Precisamos entender como o arquivo index.html se comunica com o arquivo style.css e vice-versa.

Notem como as coisas começam a se encaixar - Na aula 02 desse curso eu falei sobre a Tag <HEAD> que é o local onde colocamos as coisas que estão trabalhando nos bastidores (Nós não vemos) mas tem um papel importantíssimo na nossa estrutura, pois bem - É aqui dentro da Tag <HEAD> que iremos digitar o caminho que faz com que o index.html se comunique com o style.css e vice-versa. Vamos ver essa estrutura logo abaixo:

```
<link rel="stylesheet" href="style.css">
```

Para um melhor entendimento veremos como essa estrutura fica ao digitarmos no MS Visual Studio Code -> index.html -> Dentro da Tag <HEAD>. Conforme imagem abaixo:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Curso FrontEnd do Bem</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
</body>
</html>
```



/igor-rebolla

# - Como a Anatomia(Estrutura) do CSS é formada

O CSS fornece regras para o arquivo HTML. Cada regra permite que façamos a estilização toda da estrutura ou apenas em determinado(s) elementos (parte da estrutura).

Uma estrutura básica do CSS é formada por:

**Seletor e Declarações (Que contém Propriedade e Valor).**

Partiu ver um exemplo para fixar melhor o entendimento.

Vamos alterar a fonte e a cor de uma Tag `<p>`. Para isso usaremos:

```
p {  
    font-size: 30px;  
    color: orange;  
}
```

- **p** - é o seletor. Aqui selecionamos o p.
- **font-size** - é a declaração que contém a propriedade (font-size) e o valor que atribuímos é (30px).
- **color** - é a declaração que contém a propriedade (color) e o valor que atribuímos é (orange).

Vamos fazer uma comparação agora, criando uma TAG `<p>` no `index.html`, salvaremos esse arquivo e abriremos no navegador.

Esse é o conteúdo da TAG `<p>` que iremos digitar:

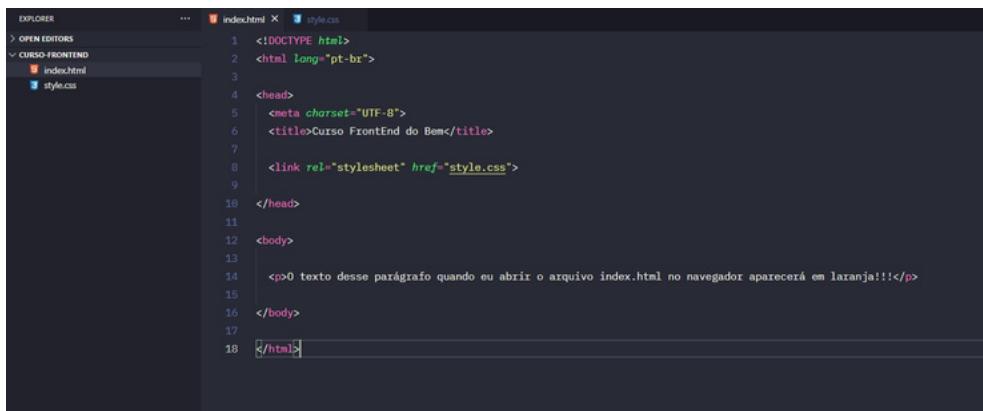
```
<p>O texto desse parágrafo quando eu abrir o arquivo index.html no navegador aparecerá em laranja!!!<p>
```



/igor-rebolla

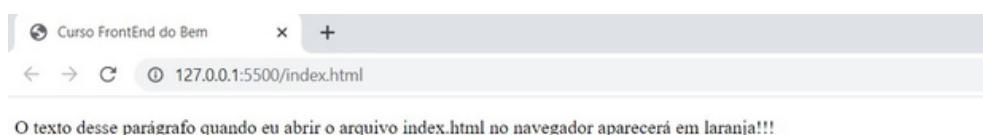
# - Como a Anatomia(Estrutura) do CSS é formada

A Tag <p> ficará assim dentro do Microsoft Visual Studio Code:



```
EXPLORER OPEN EDITORS ... index.html style.css
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>Curso FrontEnd do Bem</title>
7
8    <link rel="stylesheet" href="style.css">
9
10 </head>
11
12 <body>
13
14    <p>O texto desse parágrafo quando eu abrir o arquivo index.html no navegador aparecerá em laranja!!!</p>
15
16 </body>
17
18 </html>
```

Vamos abrir o navegador agora e ver se a Tag <p> digitada apareceu em laranja. Conforme imagem abaixo:



O texto da Tag <p> foi exibido na cor preta (pois essa é a cor padrão que aparece quando renderizamos uma Tag HTML no Navegador). Conforme vimos nas 5 aulas anteriores. Para que a cor ORANGE seja aplicada é necessário digitarmos o código da estrutura do p dentro do arquivo style.css

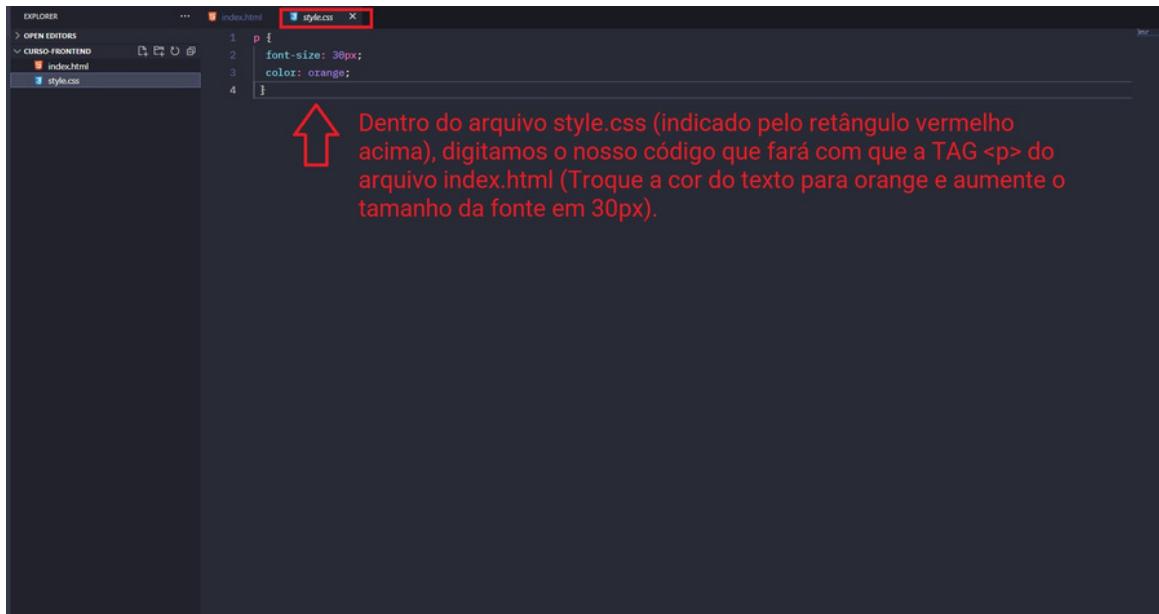


/igor-rebolla

# - Como a Anatomia(Estrutura) do CSS é formada

Agora vamos manter o que foi digitado dentro do arquivo index.html e digitaremos o código abaixo dentro do arquivo style.css

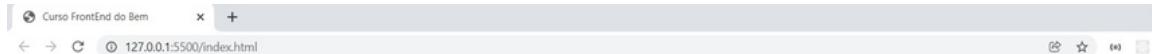
```
p {  
    font-size: 30px;  
    color: orange;  
}
```



Código digitado dentro do arquivo style.css, vamos ver na imagem da próxima página se o texto dentro da TAG <p> o qual definimos no arquivo index.html, teve sua cor alterada para (orange) e o tamanho da fonte para 30px.



# - Como a Anatomia(Estrutura) do CSS é formada



O texto desse parágrafo quando eu abrir o arquivo index.html no navegador aparecerá em laranja!!!



Agora que o código do nosso arquivo style.css foi aplicado na TAG <p> do arquivo index.html (o texto foi estilizado na cor orange-laranja e teve o tamanho da sua fonte alterada para 30px conforme definimos).

Conforme imagem acima o texto do nosso parágrafo e o tamanho da fonte foram alterados com êxito.

**É importante que esse conceito fique bem claro, pois essa é a base da base do CSS.**

Partiu fazer mais um exemplo, só que agora além da TAG <p> que já foi digitada, vamos acrescentar:

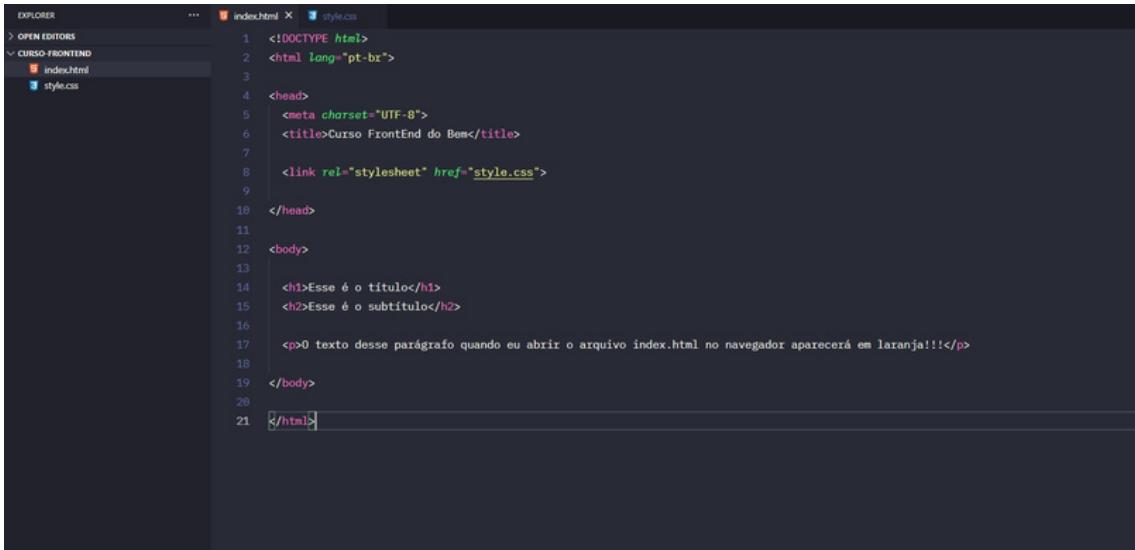
uma TAG <h1> que representa o titulo principal  
uma TAG <h2> que representa o subtítulo

Vamos ver como essa estrutura ficará no MS Visual Studio Code dentro do arquivo index.html, conforme imagem da próxima página.



/igor-rebolla

# - Como a Anatomia(Estrutura) do CSS é formada



The screenshot shows a code editor interface with two tabs open: 'index.html' and 'style.css'. The 'index.html' tab contains the following HTML code:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Curso FrontEnd do Bem</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<h1>Esse é o título</h1>
<h2>Esse é o subtítulo</h2>
<p>O texto desse parágrafo quando eu abrir o arquivo index.html no navegador aparecerá em laranja!!!</p>
</body>
</html>
```

Vamos ver o resultado do código acima mostrado no navegador. Conforme imagem abaixo:



/igor-rebolla

# - Como a Anatomia(Estrutura) do CSS é formada

Ok, Igor.... eu entendi que a TAG <p> está em laranja e com a fonte maior porque foi estilizada pelo código do arquivo style.css e qual o motivo que as TAGS <h1> e <h2> respectivamente não receberam também esse estilo?

Eu respondo: Definimos apenas o estilo para a TAG <p>. As outras 2 TAGS estão "isoladas dentro do arquivo index.html" sem qualquer vínculo direto com o arquivo style.css. E como resolvemos isso?

Dá mesma forma que fizemos com a TAG <p>, por isso eu coloquei na página 8 dessa aula um texto em vermelho dizendo a importância de entender esse conceito.

Bora então estilizar as TAGS <h1> e <h2>

Foi visto nessa aula que a Anatomia (estrutura do CSS) é formada por:

**Seletor e Declarações**, que contém **Propriedade e Valor**. E com base nessa informação (definimos duas propriedades para a TAG <p>)

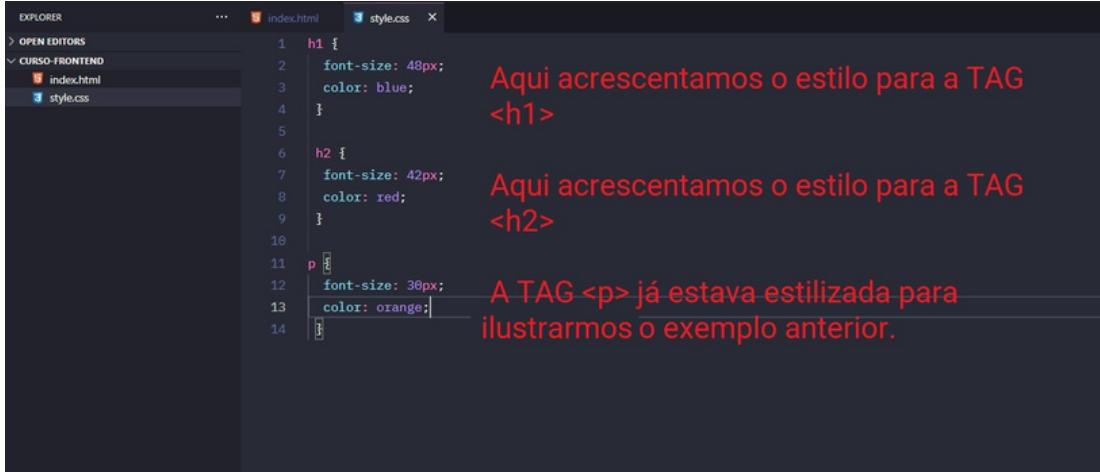
```
h1 {  
    font-size: 48px;  
    color: blue;  
}
```

```
h2 {  
    font-size: 42px;  
    color: red;  
}
```



# - Como a Anatomia(Estrutura) do CSS é formada

Vamos ver como o código definido na página anterior ficará dentro do arquivo style.css



```
EXPLORER          index.html    style.css
OPEN EDITORS
CURSO-FRONTEND
  index.html
  style.css

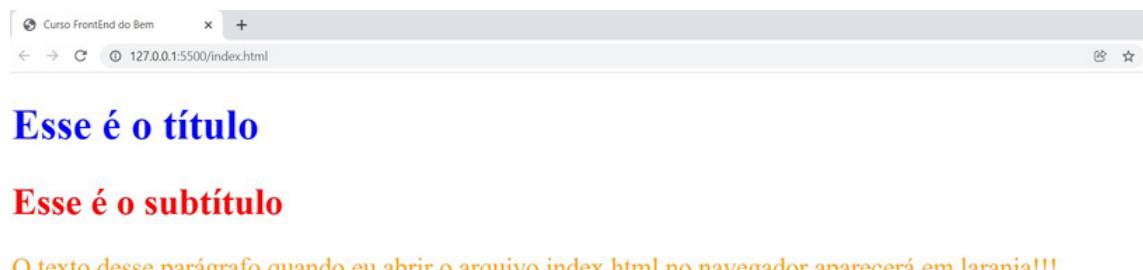
1 h1 {
2   font-size: 48px;
3   color: blue;
4 }
5
6 h2 {
7   font-size: 42px;
8   color: red;
9 }
10
11 p {
12   font-size: 30px;
13   color: orange;
14 }
```

Aqui acrescentamos o estilo para a TAG <h1>

Aqui acrescentamos o estilo para a TAG <h2>

A TAG <p> já estava estilizada para ilustrarmos o exemplo anterior.

Vamos ver o resultado do código digitado acima sendo mostrado no navegador. Conforme imagem abaixo:



Conforme imagem acima o cor do texto e o tamanho da fonte tanto do <h1> quanto do <h2> foram alterados com êxito.



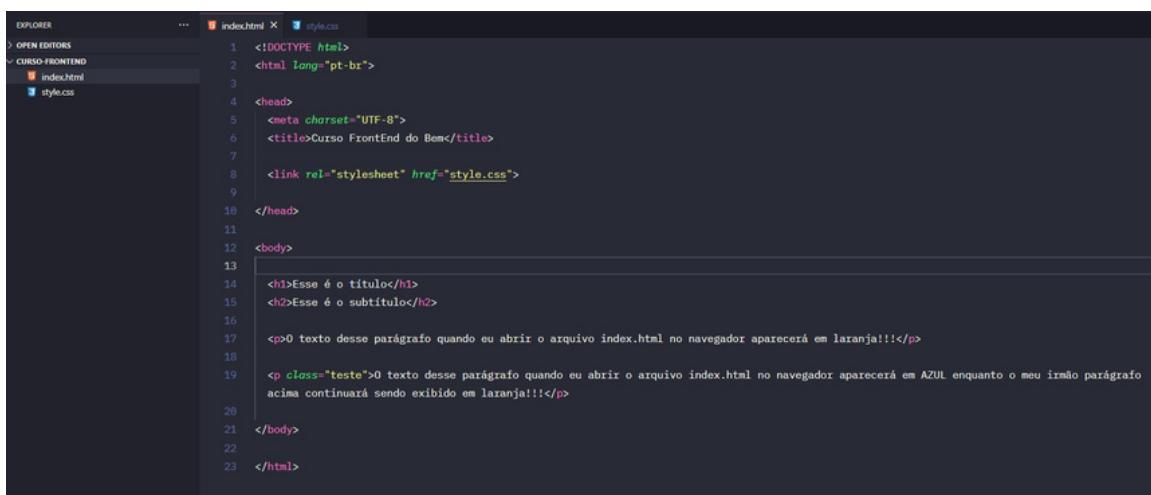
# - Como a Anatomia(Estrutura) do CSS é formada

E se eu quiser acrescentar mais uma TAG <p> no meu arquivo index.html e o texto dessa nova TAG ser modificado para azul e o tamanho da fonte para 50px.

Eu sei que já temos uma TAG <p> que está estilizada em orange e o tamanho da sua fonte está em 30px. É possivel fazer isso sem gerar conflito com a primeira TAG <p>?

Sim, é possível!!! Na aula 5 (anterior a essa) vimos o conceito de **CLASSES**, pois bem é com ela que resolveremos a situação proposta acima. Bora ver como:

Primeiro passo é irmos no arquivo index.html e digitarmos mais uma TAG <p> só que dessa vez vamos acrescentar uma classe ( representada pela palavra **class**) logo após a TAG de abertura <p> e um nome (aqui o nome escolhido foi "**teste**") para que o arquivo style.css possa identificar essa classe e aplicar os estilos definidos.



The screenshot shows a code editor interface with two files open: index.html and style.css. The index.html file contains the following HTML code:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Curso FrontEnd do Bem</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Esse é o título</h1>
    <h2>Esse é o subtítulo</h2>
    <p>O texto desse parágrafo quando eu abrir o arquivo index.html no navegador aparecerá em laranja!!!</p>
    <p class="teste">O texto desse parágrafo quando eu abrir o arquivo index.html no navegador aparecerá em AZUL enquanto o meu irmão parágrafo acima continuará sendo exibido em laranja!!!</p>
</body>
</html>
```

The style.css file contains the following CSS code:

```
.teste {
    color: blue;
}
```



/igor-rebolla

# - Como a Anatomia(Estrutura) do CSS é formada

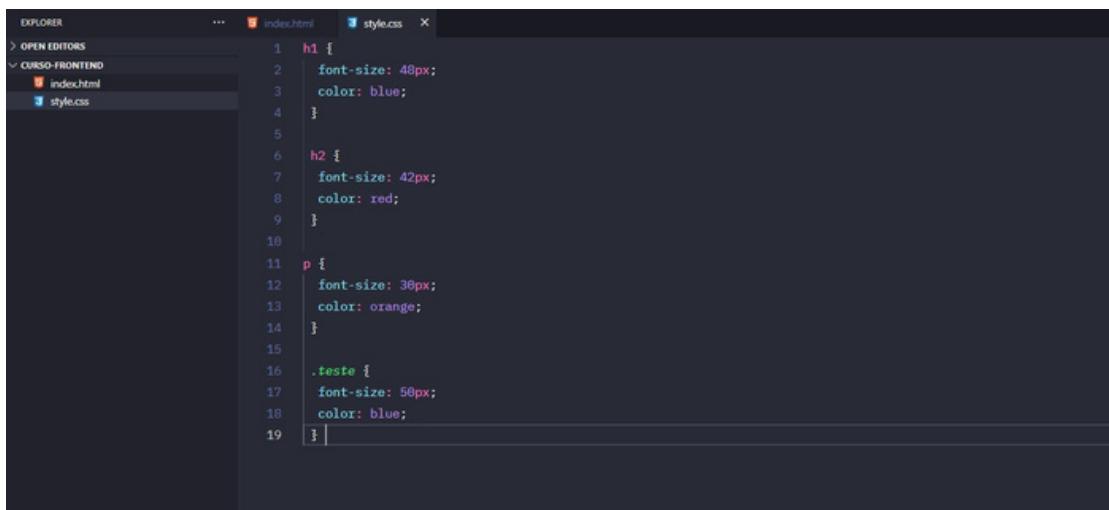
Na imagem abaixo vamos digitar o código dentro do arquivo style.css para estilizarmos a classe do nosso segundo parágrafo o qual demos o nome aqui de "teste".

A sintaxe da Classe dentro do arquivo style.css é representada por:

**. nome da classe**

No nosso exemplo abaixo ficou assim:

**.teste**

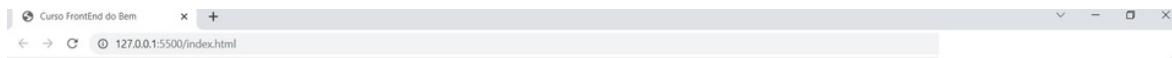


```
1  h1 {  
2    font-size: 48px;  
3    color: blue;  
4  }  
5  
6  h2 {  
7    font-size: 42px;  
8    color: red;  
9  }  
10  
11 p {  
12   font-size: 36px;  
13   color: orange;  
14 }  
15  
16 .teste {  
17   font-size: 56px;  
18   color: blue;  
19 }
```

Com os códigos devidamente digitados tanto no arquivo index.html quanto no arquivo style.css, vamos ver o resultado exibido no navegador conforme imagem da próxima página.



# - Como a Anatomia(Estrutura) do CSS é formada



**Esse é o título**

**Esse é o subtítulo**

O texto desse parágrafo quando eu abrir o arquivo index.html no navegador aparecerá em laranja!!!

O texto desse parágrafo quando eu abrir o arquivo index.html no navegador aparecerá em AZUL enquanto o meu irmão parágrafo acima continuará sendo exibido em laranja!!!

Conforme imagem acima por conta da classe teste na segunda TAG `<p>` o cor do texto e o tamanho da fonte foram alterados com êxito e a primeira TAG `<p>` manteve o seu estilo sem sofrer qualquer alteração.

# Sugestões de prática para essa aula:

1. Abrir o Microsoft Visual Studio Code, depois a pasta Curso-FrontEnd e criar um arquivo "style.css" conforme demonstrado nessa aula
2. Digitar dentro da TAG <head> em index.html a estrutura que faz com que o HTML "converse" com o CSS. Vou deixá-la aqui embaixo:  
`<link rel="stylesheet" href="style.css">`
3. Digitar os exemplos dessa aula, salvar e visualizar no navegador
4. Faça um novo exemplo:
  - alterando a cor da tag <p> para **green** e o texto onde está **laranja** para **verde**.
  - alterar o nome da classe teste para teste001 no arquivo index.html, alterar também o nome da classe no arquivo style.css para .teste001 a sua cor para (**red**) e o tamanho da texto ( font-size: 36px)



/igor-rebolla

# O que é uma Box Model

Chegamos na segunda aula do Segundo pilar do nosso curso que é o CSS.

Igor, e o que é esse tal de Box Model?

Os Sites eles são criados em blocos. Todo site é formado por pequenos blocos que vão se encaixando e assim criando o seu layout final.

Sendo assim esse modelo de caixas (Box Model) é o que vai definir o tamanho desses blocos, qual a distância de um para outro e como funciona a altura e a largura.

Vamos usar a página inicial do Linkedin como exemplo para visualizarmos como esses blocos estão distribuídos:

1. O Bloco Número 1 representa o menu de login da página
2. O Bloco Número 2 é o container (envolve) tanto o bloco 2.1 (um texto) quanto o bloco 2.2 (uma imagem)
3. O Bloco Número 3 é o container (envolve) tanto o bloco 3.1 (um texto) quanto o bloco 3.2
4. O Bloco 4 está fazendo referência a um texto e um botão, mas nada impede de definirmos aqui também um bloco para o texto o qual chamaríamos de bloco 4.1 e um bloco para o botão o qual chamaríamos de bloco 4.2.

Vamos ver a distribuição desses blocos na imagem da próxima página:



# O que é uma Box Model



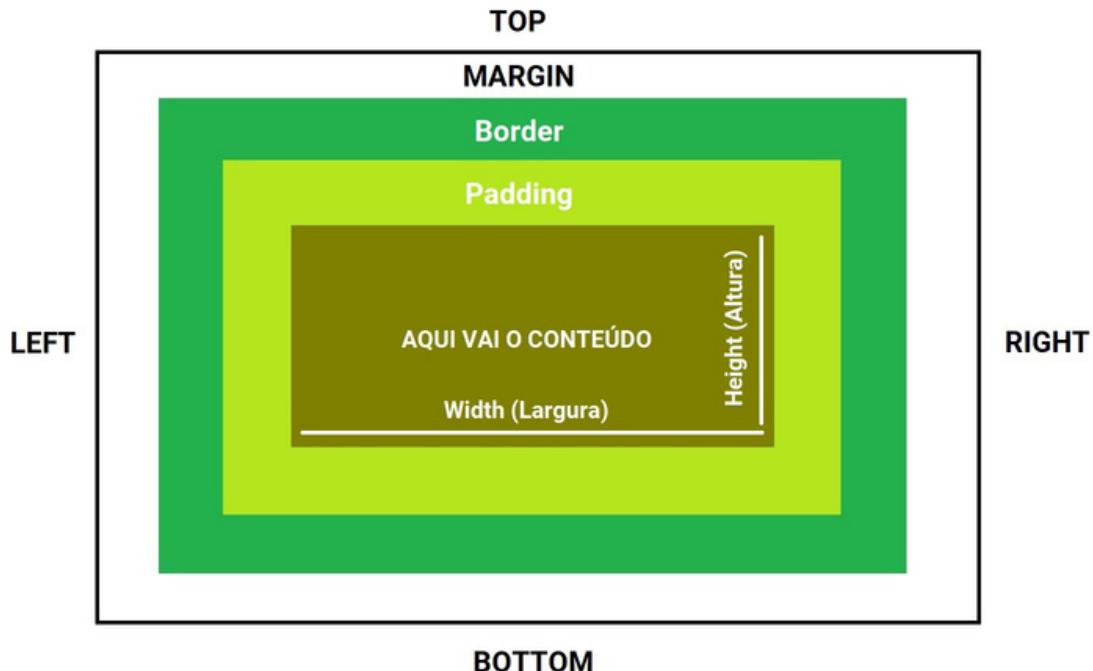
Resumindo: Tudo isso são caixas (boxes) que vamos montando e assim o layout do site vai sendo criado.

O importante aqui é entender como essa caixas funcionam para podermos manipula-las e chegar ao resultado final esperado (seja para o nosso estudo, portfólio ou para o layout do site do cliente que foi especificado).

Na próxima página eu vou passar o meu entendimento seguindo o exemplo tradicional visto na internet (acredito que seja importante ter o contato com esse exemplo). Bora ver esse exemplo então:



# O que é uma Box Model



## **Content (Conteúdo)**

O Content é o que define a largura inicial da caixa

## **Padding (Preenchimento)**

O Padding é o que separa o conteúdo das bordas da caixa. Resumindo é a margin (margem) interna.

## **Border (Borda)**

O Border como o próprio nome sugere é o que vai definir bordas para a caixa.

## **Margin (Margem)**

A Margin é o que vai definir a distância entre uma caixa e outra caixa. Resumindo é a margin (margem) externa.

# O que é uma Box Model

Começamos nossa estrutura sempre pelo Content (Conteúdo). Aqui o exemplo está feito em texto com a frase (Aqui vai o Conteúdo), mas pode ser aplicado em uma imagem também e etc...

O conteúdo vai ter uma caixa com um tamanho de largura (width) e altura (height)

A partir do conteúdo vem um carinha chamado Padding, o Padding seria exatamente o distanciamento/espaçamento entre o conteúdo e a borda.

A partir do Padding nós temos a Borda (Border), onde eu posso aplicar ou não um valor para essa border. Por exemplo: A Borda vai ter 15px de largura e ele vai definir para todos os cantos aqui definido como (left, top, right e bottom).

Uma coisa muito importante para termos em mente é a seguinte:

A Soma do total da largura do conteúdo + Padding + Border por padrão vai ser o total da largura desse elemento. E o total da largura aqui chamamos de Width (largura).

Já a soma do Border-Top + Padding-Top + Content + Padding-Bottom e Margin-Bottom vai ser o total da altura desse elemento. E o total da altura aqui chamamos de Height (altura).

Por que o exemplo da construção de uma casa passa a fazer sentido tomando como base os 2 parágrafos acima, principalmente quando falamos Width (largura) do terreno. Vamos entender melhor na próxima página. Partiu ver....



# O que é uma Box Model

Imagina que nós temos um terreno com 10metros de largura (vamos focar aqui agora) por 10metros de comprimento. E queremos que a nossa sala tenha exatos 10metros de largura contando as paredes. Pra facilitar nossa conta aqui, vamos definir a largura do bloco da parede da esquerda em 1metro e o a largura do bloco da parede da direita em 1metro.

Ou seja: Com base no que foi definido na situação acima, a nossa sala terá uma largura de 8m e as paredes uma largura total de 2m.

Agora se você tem um vizinho do lado esquerdo que é brother e não liga que você use a parede dele (o famoso germinado) de um lado ai sua sala ficaria com 9m e só a parede do lado direito seria contabilizada, totalizando 1 metro.

O mesmo vale para o vizinho do lado direito.



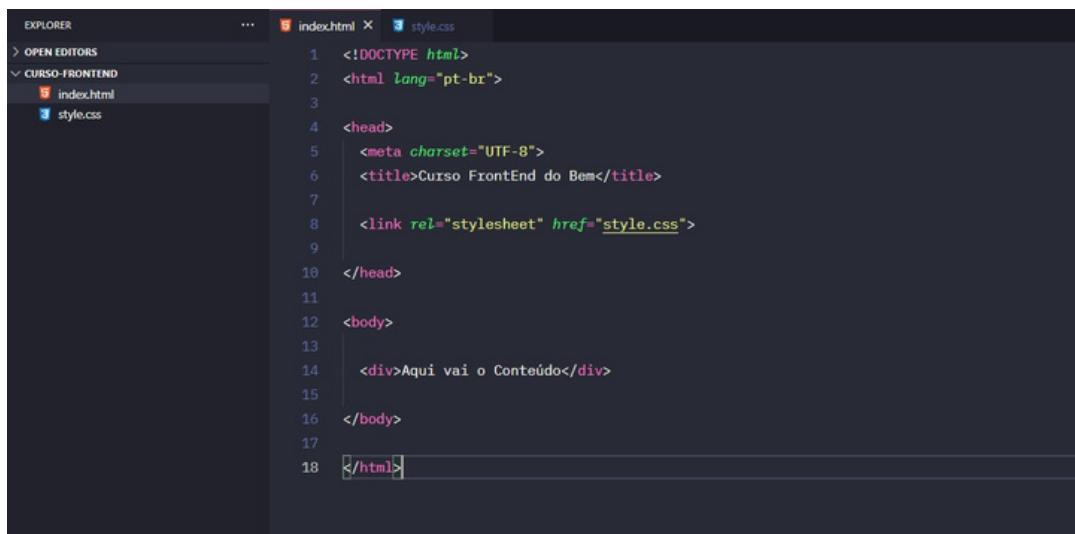
# Exemplos sobre Box Model

Agora vamos estruturar o nosso código para digitarmos dentro do Microsoft Visual Studio Code e vermos na prática como o Box Model funciona.

Vamos aproveitar a estrutura criada nas aulas anteriores:

- 1 - Abrir Microsoft Visual Studio Code;
- 2 - Abrir a pasta Curso-FrontEnd;
- 3 - Abrir o arquivo index.html;
- 4 - Abrir o arquivo style.css

Dentro do arquivo index.html, vamos criar uma `<div></div>` dentro de body. Lembrando que a `<div>` cria um elemento de bloco.



```
EXPLORER ... index.html x style.css
OPEN EDITORS
CURSO-FRONTEND
index.html
style.css

1 <!DOCTYPE html>
2 <html lang="pt-br">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>Curso FrontEnd do Bem</title>
7
8   <link rel="stylesheet" href="style.css">
9
10 </head>
11
12 <body>
13
14   <div>Aqui vai o Conteúdo</div>
15
16 </body>
17
18 </html>
```

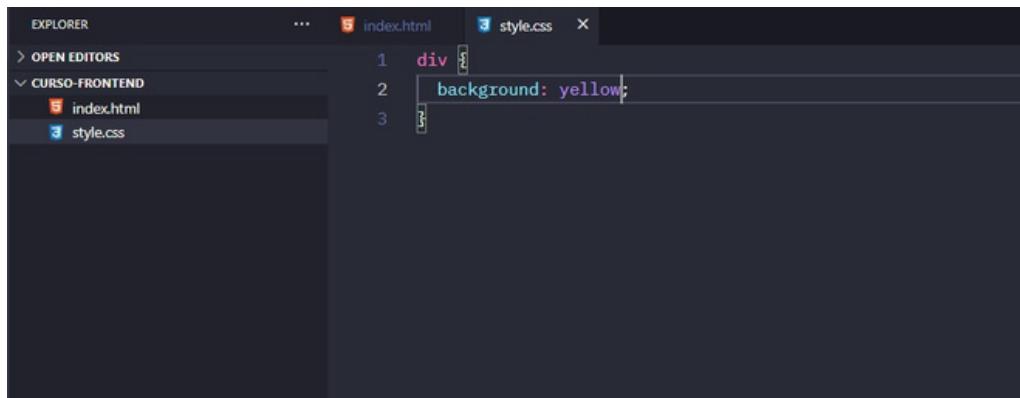


# Exemplos sobre Box Model

Dentro do arquivo style.css, vamos selecionar essa <div> e colocar um background (cor de fundo) assim poderemos ver o que está acontecendo na <div>.

O código vai ficar assim:

```
div {  
    background: yellow;  
}
```



Ao abrir o código index.html no navegador eu tenho o resultado conforme imagem abaixo:



# Exemplos sobre Box Model

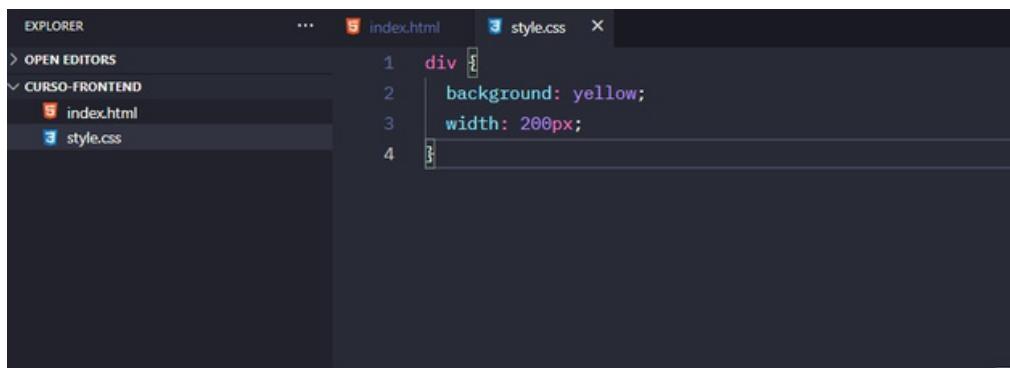
E qual o motivo do yellow (amarelo) ter tomado conta da largura total da tela?

Quando especificamos o background em yellow dentro do arquivo style.css (não definimos nenhuma largura (width)) para esse conteúdo e por padrão ele assumiu o valor auto (automático).

Vamos definir um valor manual para o width de 200px.

Agora no nosso arquivo style.css (além do background, teremos o width definido)

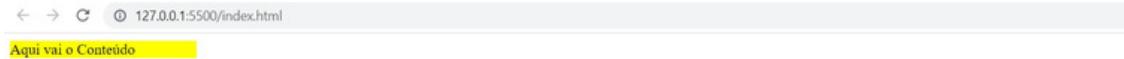
```
div {  
    background: yellow;  
    width: 200px;  
}
```



Vamos ver na imagem da próxima página como ficou no navegador com essa alteração:



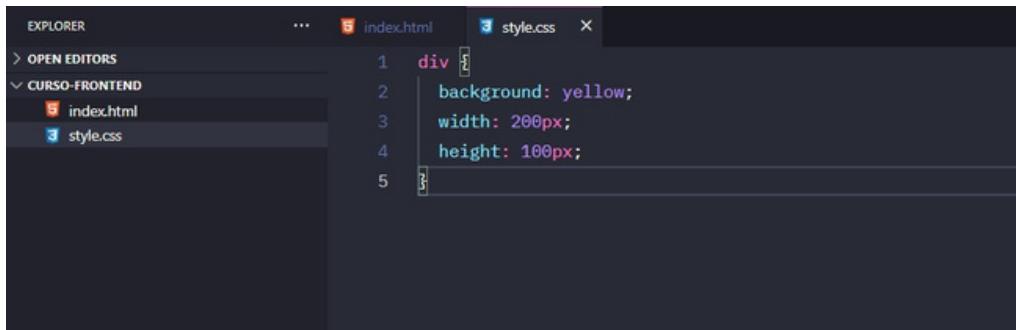
# Exemplos sobre Box Model



Notem que o yellow já não toma mais conta da largura total da tela e agora está mostrando os 200px que definimos no arquivo style.css.

Agora vamos acrescentar uma altura (Height)

```
div {  
    background: yellow;  
    width: 200px;  
    height: 100px;  
}
```



Vamos ver na próxima página como ficou no navegador acrescentando o height (altura) de 100px:



# Exemplos sobre Box Model

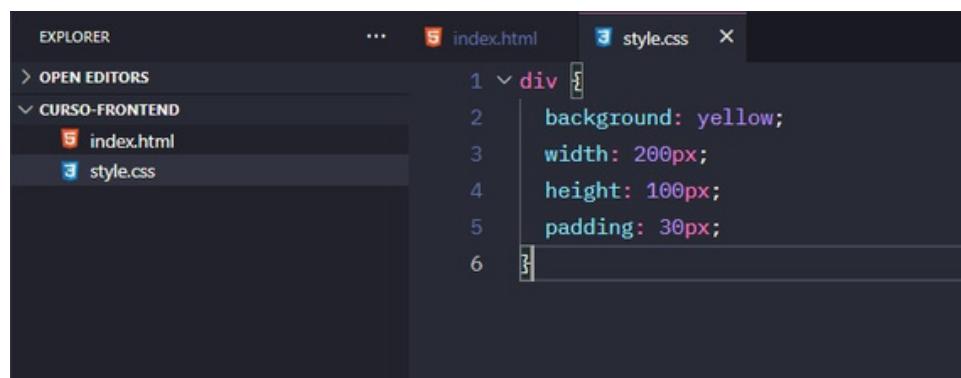
← → ⌂ ⓘ 127.0.0.1:5500/index.html

Aqui vai o Conteúdo

Agora além do background (yellow), width (200px) temos também um height de (100px).

O próximo elemento que iremos acrescentar aqui é o padding (lembrem que o padding é a margem (margin) interna)).

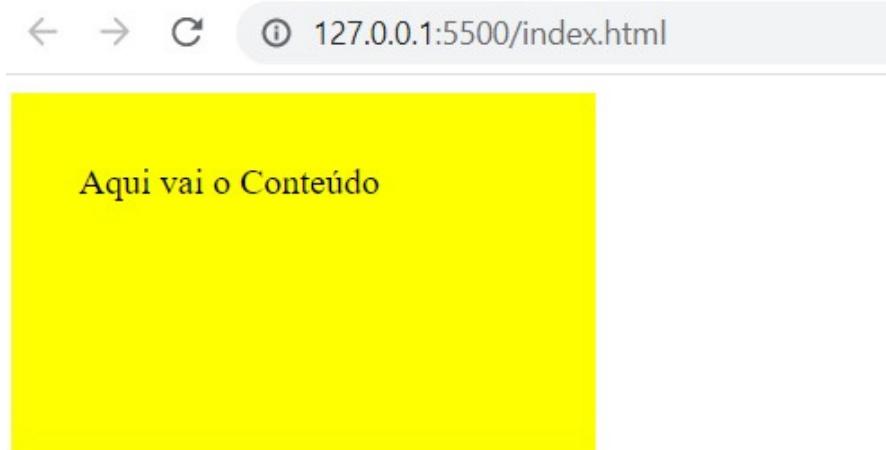
```
div {  
    background: yellow;  
    width: 200px;  
    height: 100px;  
    padding: 30px;  
}
```



/igor-rebolla

# Exemplos sobre Box Model

Vamos ver como ficou no navegador acrescentando o padding (margem interna) de 30px:



Podemos notar que o padding de 30px por ser uma margem interna entre o "Aqui vai o Conteúdo" e a ponta da esquerda e entre o "Aqui vai o Conteúdo" e a parte de cima.

Agora além do background (yellow), width (200px), height (100px), temos também um padding (30px).

O próximo elemento que iremos acrescentar aqui é o border (borda).

Antes de adicionarmos o border, vamos entender como funciona sua estrutura.



# Exemplos sobre Box Model

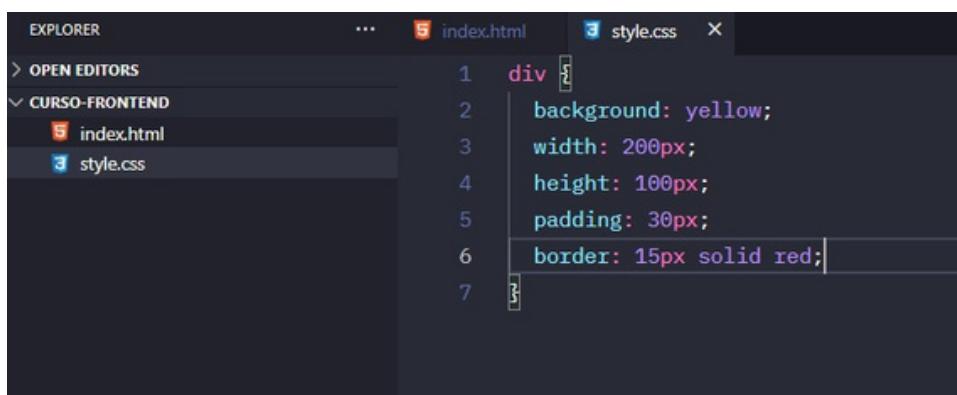
border: 15px solid red

1    2    3    4

- 1 - border é o nome do elemento o qual queremos definir
- 2 - o tamanho da borda (aqui definimos 15px)
- 3 - o tipo da borda (aqui escolhemos solid)
- 4 - a cor dessa borda (aqui escolhemos red)

Agora sim podemos acrescentar o border em nosso código.

```
div {  
background: yellow;  
width: 200px;  
height: 100px;  
padding: 30px;  
border: 15px solid red  
}
```

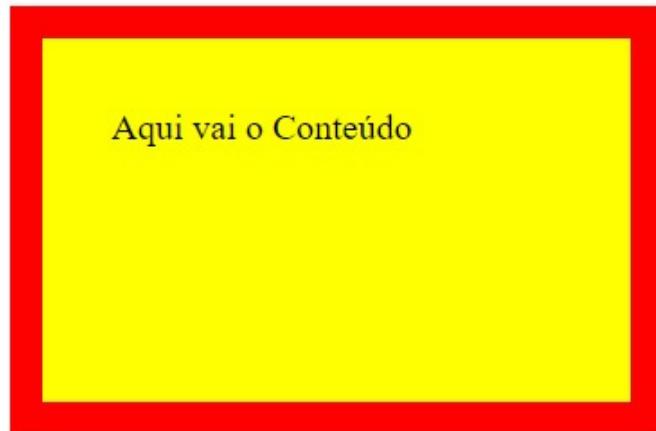


Vamos ver na próxima página como ficou no navegador acrescentando a (border: 15px solid red)



# Exemplos sobre Box Model

← → ⌂ 127.0.0.1:5500/index.html



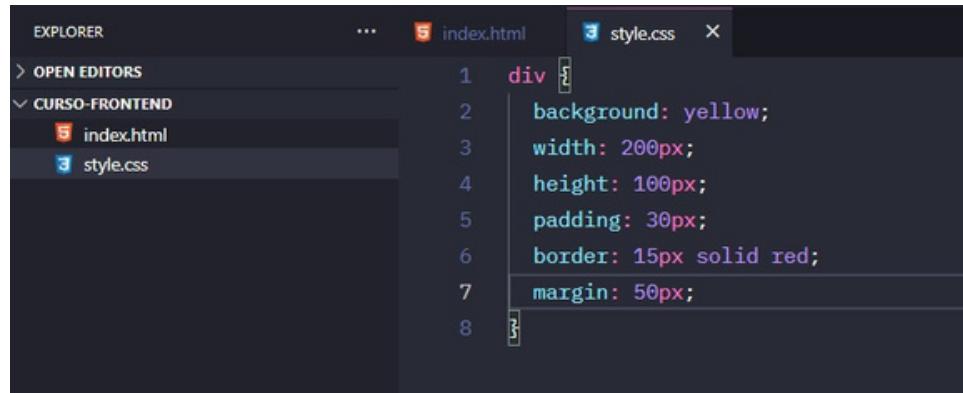
Agora além do background (yellow), width (200px), height (100px), padding (30px), temos também uma border (15px solid red).

O próximo elemento que iremos acrescentar aqui é a margin (Margem).

```
div {  
    background: yellow;  
    width: 200px;  
    height: 100px;  
    padding: 30px;  
    border: 15px solid red;  
    margin: 50px;  
}
```



# Exemplos sobre Box Model



```
EXPLORER ... 5 index.html style.css
> OPEN EDITORS
CURSO-FRONTEND
  index.html
  style.css

1 div {
2   background: yellow;
3   width: 200px;
4   height: 100px;
5   padding: 30px;
6   border: 15px solid red;
7   margin: 50px;
8 }
```

Vamos ver como ficou no navegador acrescentando a margin de 50px:



Podemos notar que foi aplicado uma margin de 50px tanto na esquerda quanto na parte de cima.

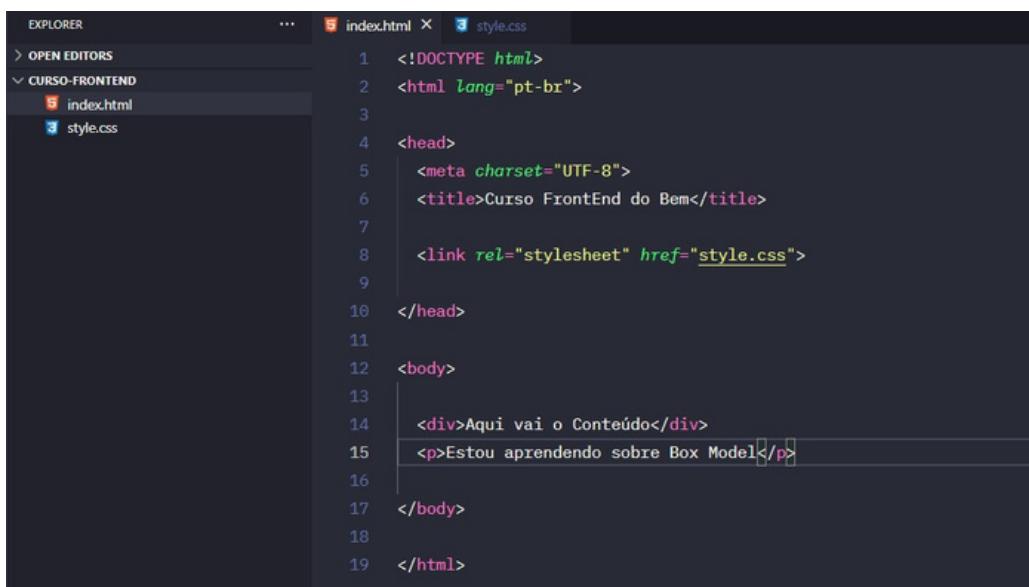


# Exemplos sobre Box Model

Para um melhor entendimento vamos definir agora outro elemento.

Vamos no index.html e colocar um elemento:

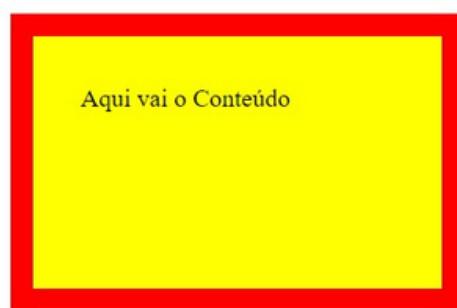
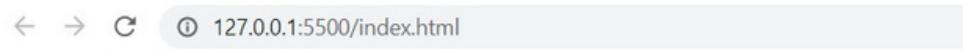
```
<p>Estou aprendendo sobre Box Model</p>
```



```
EXPLORER          ...  index.html ×  style.css
OPEN EDITORS
CURSO-FRONTEND
index.html
style.css

1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>Curso FrontEnd do Bem</title>
7
8    <link rel="stylesheet" href="style.css">
9
10 </head>
11
12 <body>
13
14  <div>Aqui vai o Conteúdo</div>
15  <p>Estou aprendendo sobre Box Model</p>
16
17 </body>
18
19 </html>
```

Se salvarmos e verificarmos como esse `<p>` ficou no navegador, o resultado será esse:



Estou aprendendo sobre Box Model



/igor-rebolla

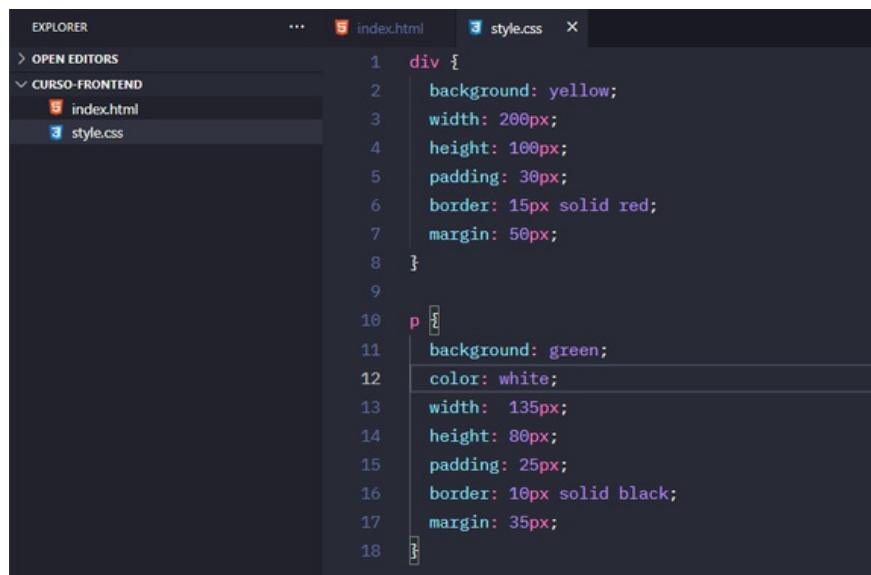
# Exemplos sobre Box Model

Notem que o <p> não sofreu nenhuma alteração. Ele está sendo mostrado com o seu tamanho(padrão) e a cor (padrão) também.

Assim como fizemos com a <div> vamos definir alguns valores também para o <p>. E isso será feito logo abaixo o fechamento da declaração da } no arquivo style.css

```
p {  
    background: green;  
    color: white;  
    width: 135px;  
    height: 80px;  
    padding: 25px;  
    border: 10px solid black;  
    margin: 35px;  
}
```

obs.: No código acima eu acrescentei (color: white) propriedade o qual vimos na aula 06 desse curso.



The screenshot shows a code editor interface with two tabs: 'index.html' and 'style.css'. The 'style.css' tab is active, displaying the following CSS code:

```
div {  
    background: yellow;  
    width: 200px;  
    height: 100px;  
    padding: 30px;  
    border: 15px solid red;  
    margin: 50px;  
}  
  
p {  
    background: green;  
    color: white;  
    width: 135px;  
    height: 80px;  
    padding: 25px;  
    border: 10px solid black;  
    margin: 35px;  
}
```



/igor-rebolla

# Exemplos sobre Box Model

Se salvarmos e verificarmos como esse `<p>` ficou no navegador depois das alterações, o resultado será esse:



Definimos e exemplificamos elemento por elemento com base na imagem localizada na página 4 desta aula. Aplicamos um background que no caso foi o yellow, depois viemos seguindo sua estrutura:

- Definimos o conteúdo;
- Definimos sua largura;
- Definimos sua altura;
- Definimos o padding;
- Definimos a border;
- Definimos a margin;

# Eu sou uma Box Model

Bom agora que vimos:

- as caixas (boxes) no exemplo da página inicial do Linkedin;
- o meu entendimento com base no exemplo padrão;
- exemplo criando uma <div> e um <p>

Vou deixar o último exemplo aqui abaixo:

Prometo que na próxima aula eu volto a pensar fora da caixa ou do guarda-roupas nesse caso. 😊😊😊

Eu sou uma box model - Onde:

- Eu estou representando o conteúdo;
- O padding que é a margin interna representa o espaçamento interno entre a minha pessoa e a borda superior (representada pela linha vermelha);
- E a margin é a parte externa entre a box model e o a estrutura do guarda-roupa.



# Eu sou uma Box Model



/igor-rebolla

# **Sugestões de prática para essa aula:**

1. Abrir o Microsoft Visual Studio Code, depois a pasta Curso-FrontEnd e os arquivos: index.html e style.css;
2. Refazer os exemplos dessa aula - da página 07 até a 18 desse pdf. Seguindo o passo a passo. Salvar e visualizar no navegador
3. Faça um novo exemplo, criando:
  - Uma <div></div>;
  - Um <h1></h1>
  - Um <p></p>
  - Salvar e visualizar no navegador

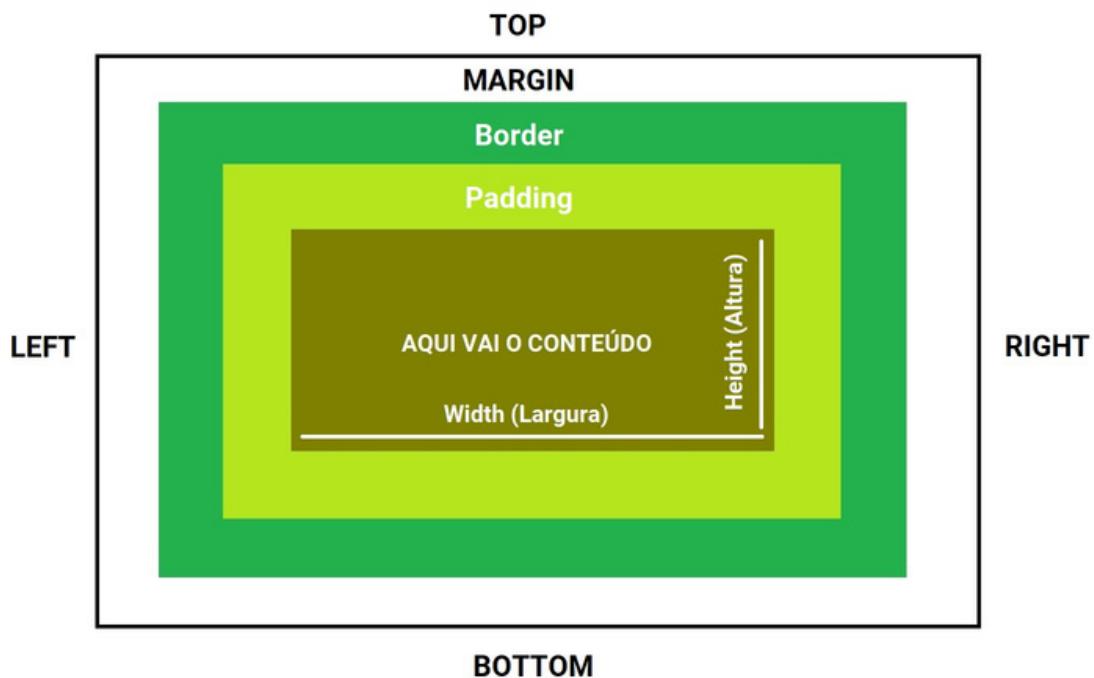


**/igor-rebolla**

# Padding e Margin (Reforçando)

Vamos aproveitar para reforçar os conceitos de Padding e Margin os quais vimos na aula 07 desse curso.

"Tá eu queria ver conteúdo novo. Porque estamos reforçando isso?" É importantíssimo o entendimento das Vossas estruturas (anatomia) e como o padding e a margin são aplicados em nossos projetos.



## Padding (Preenchimento)

O Padding é o que separa o conteúdo das bordas da caixa. Resumindo é a margin (margem) interna.

## Margin (Margem)

A Margin é o que vai definir a distância entre uma caixa e outra caixa. Resumindo é a margin (margem) externa.



# Padding e Margin (Reforçando)

Na figura da página anterior, notem que existem 4 palavras escritas fora da caixa (box) na parte branca: **Top, Right, Bottom e Left**

Essas quatro palavras podem ser **aplicadas** tanto para o **Padding** quanto para a **Margin**. Lembrando que Padding e Margin são usados dentro do arquivo **style.css**.

**Quase ia me esquecendo: A leitura do Top, Right, Bottom e Left (é feita no sentido horário, começando: Pelo Top, passando pelo Right, dando um oi no Bottom e uhuuu cheguei ao Left.**

Tá, Igor... você falou que eu tenho 4 palavras tanto para o padding quanto para a margin, eu sou obrigado a usar todas elas ou eu posso usar conforme minha necessidade?

Que pergunta mais bacanuda essa: Você **não** é obrigado a utilizar todas as 4 palavras (Top, Right, Bottom e Left). Vamos ver alguns exemplos na próxima página para melhor entendimento



/igor-rebolla

# Exemplos Padding e Margin

## Exemplo 1:

Eu preciso definir um padding de 30px para os 4 lados e uma margin de 40px para os 4 lados. Como eu faço isso?

Existem 2 formas:

Primeira Forma do Exemplo 1 (para didática e aprendizado)

padding: 30px 30px 30px 30px  
1      2      3      4

Onde:

- 1 - padding-top: 30px
- 2 - padding-right: 30px
- 3 - padding-bottom: 30px
- 4 - padding-left: 30px

margin: 50px 50px 50px 50px  
1      2      3      4

- 1 - margin-top: 50px
- 2 - margin-right: 50px
- 3 - margin-bottom: 50px
- 4 - margin-left: 50px

Segunda Forma (utilizada no dia a dia)

padding: 30px (Digitar 30px uma única vez faz com que o CSS atribua o valor de 30px para os 4 lados)  
margin: 50px (Digitar 50px uma única vez faz com que o CSS atribua o valor de 50px para os 4 lados)



# Exemplos Padding e Margin

## Exemplo 2:

Agora eu tenho algo mais específico para o padding onde eu preciso definir:

- padding de 30px para o Top e 30px para o Bottom
- padding de 15px para o Right e 15px para o Left

Existem 2 formas:

Primeira Forma do Exemplo 2 (para didática e aprendizado)

padding: 30px 15px 30px 15px  
1      2      3      4

Onde:

- 1 - padding-top: 30px
- 2 - padding-right: 15px
- 3 - padding-bottom: 30px
- 4 - padding-left: 15px

## Exemplo 2:

Agora eu tenho algo mais específico para a margin onde eu preciso definir:

- margin de 40px para o Top e 40px para o Bottom
- margin de 25px para o Right e 25px para o Left

margin: 40px 25px 40px 25px  
1      2      3      4

- 1 - margin-top: 40px
- 2 - margin-right: 25px
- 3 - margin-bottom: 40px
- 4 - margin-left: 25px



/igor-rebolla

# Exemplos Padding e Margin

Exemplo 2:

Segunda Forma do Exemplo 2 (utilizada no dia a dia)

padding: 30px 15px

1    2

1 - Atribui o valor de 30px tanto para o padding-top quanto para o bottom

2 - Atribui o valor de 15px tanto para o padding-right quanto para o left

margin: 40px 25px

1    2

1 - Atribui o valor de 40px tanto para a margin-top quanto para a bottom

2 - Atribui o valor de 25px tanto para a margin-right quanto para a left

Agora que vimos a forma didática e a forma usada no dia a dia vamos ver um exemplo de cada (assim visualizamos e tiramos nossas dúvidas se existe alguma diferença quando o código do exemplo 1 for visualizado no navegador)

Forma didática:

padding: 30px 30px 30px 30px;

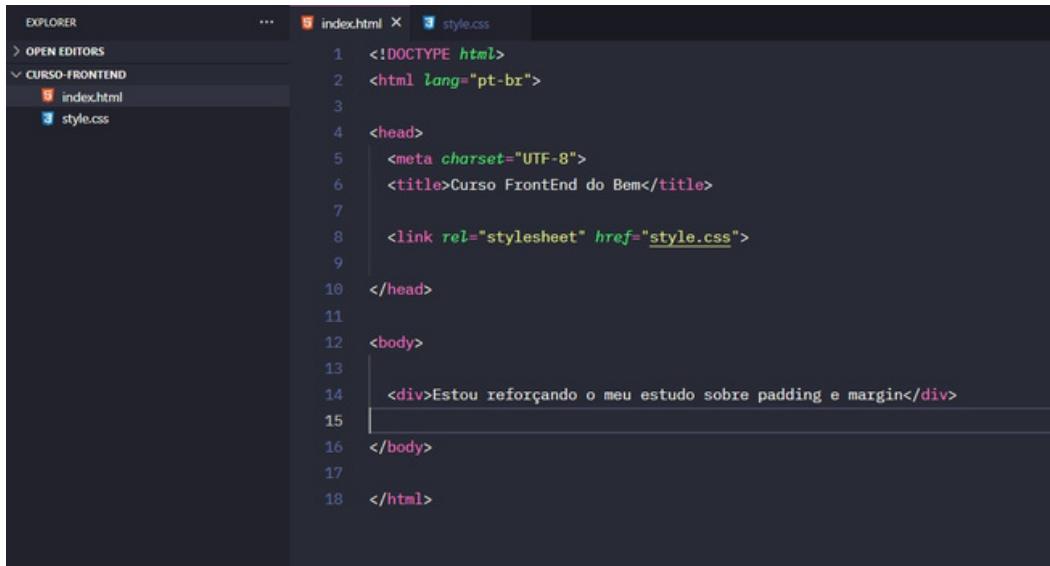
margin: 50px 50px 50px 50px;



/igor-rebolla

# Exemplos Padding e Margin

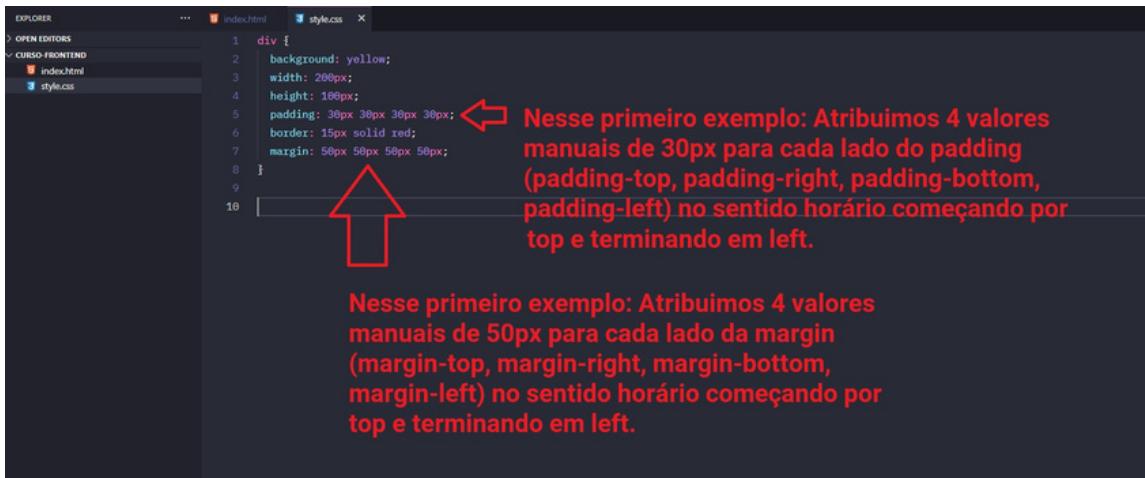
Vamos aproveitar a estrutura do arquivo index.html da aula 07 (Aqui foi alterado apenas o conteúdo entre as Divs conforme imagem abaixo (Exemplo 1 - Forma didática)):



The screenshot shows a code editor interface with two tabs open: 'index.html' and 'style.css'. The 'index.html' tab contains the following HTML code:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Curso FrontEnd do Bem</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<div>Estou reforçando o meu estudo sobre padding e margin</div>
</body>
</html>
```

Agora vamos ver abaixo como a (forma didática) foi digitada no nosso arquivo style.css:



The screenshot shows a code editor interface with two tabs open: 'index.html' and 'style.css'. The 'style.css' tab contains the following CSS code:

```
div {
background: yellow;
width: 200px;
height: 100px;
padding: 30px 30px 30px 30px; ←
border: 15px solid red;
margin: 50px 50px 50px 50px;
}
```

Two annotations are present:

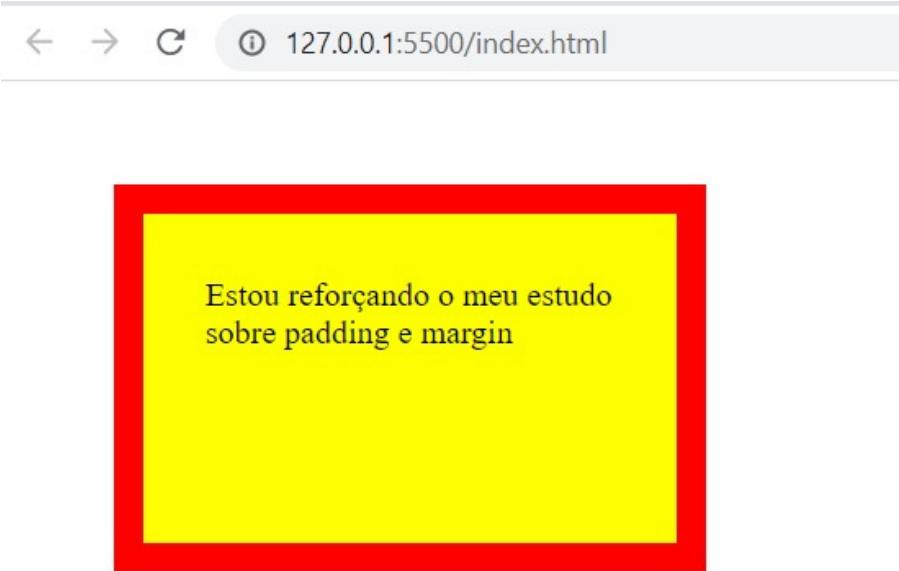
- An annotation pointing to the padding rule with the text: "Nesse primeiro exemplo: Atribuimos 4 valores manuais de 30px para cada lado do padding (padding-top, padding-right, padding-bottom, padding-left) no sentido horário começando por top e terminando em left."
- An annotation pointing to the margin rule with the text: "Nesse primeiro exemplo: Atribuimos 4 valores manuais de 50px para cada lado da margin (margin-top, margin-right, margin-bottom, margin-left) no sentido horário começando por top e terminando em left."



/igor-rebolla

# Exemplos Padding e Margin

Vamos como o exemplo didático ficou renderizado no navegador conforme imagem abaixo:



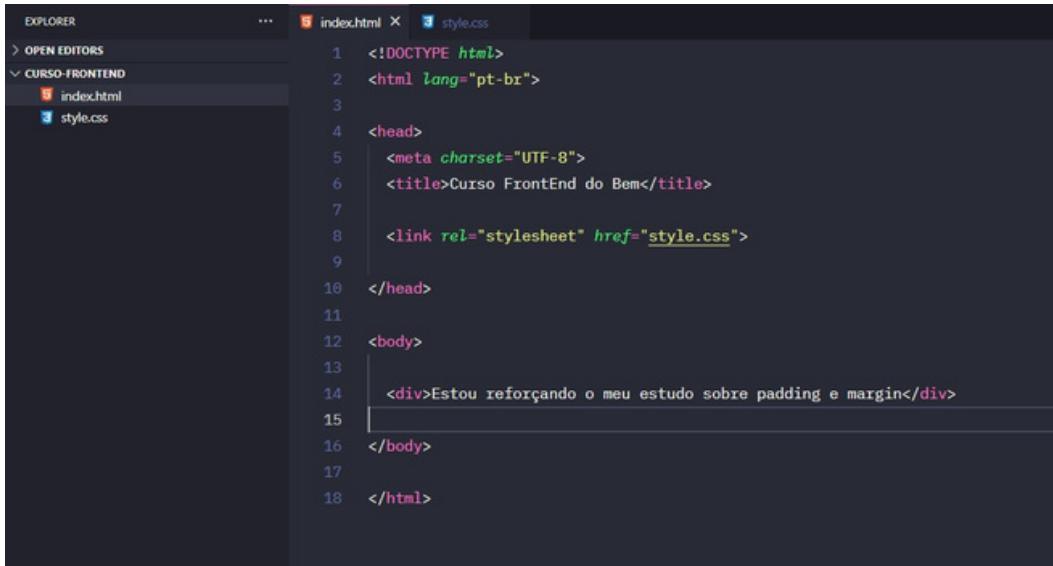
Vamos aproveitar a mesma estrutura do arquivo index.html da aula 07 e visualizar no navegador o Exemplo 01 - Só que aqui será visto da forma que é usada no dia a dia.

Forma Usada no dia a dia:

padding: 30px;  
margin: 50px;



# Exemplos Padding e Margin



The screenshot shows a code editor interface with two tabs open: 'index.html' and 'style.css'. The 'index.html' tab contains the following HTML code:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Curso FrontEnd do Bem</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<div>Estou reforçando o meu estudo sobre padding e margin</div>
</body>
</html>
```

Agora vamos ver abaixo como a (forma usada no dia a dia) foi digitada no nosso arquivo style.css:



The screenshot shows a code editor interface with the 'style.css' tab open. The CSS code is as follows:

```
div {
    background: yellow;
    width: 200px;
    height: 100px;
    padding: 30px; ← Aqui foi aplicado um padding de 30 px para os 4 lados - Top, Right, Bottom e Left (No sentido horário)
    border: 15px solid #d3d3d3;
    margin: 50px;
}
```

Annotations in red text with arrows explain the styling:

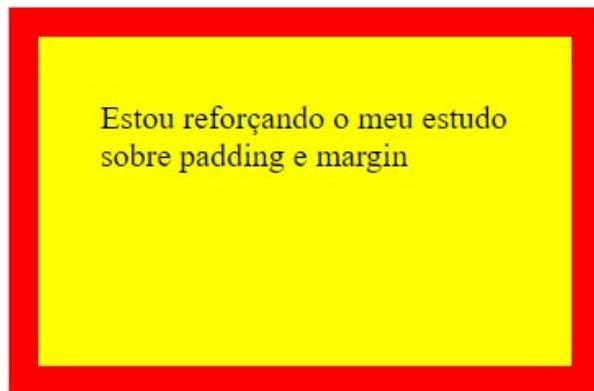
- A red arrow points to the 'padding: 30px;' line with the text: "Aqui foi aplicado um padding de 30 px para os 4 lados - Top, Right, Bottom e Left (No sentido horário)".
- A red arrow points to the 'margin: 50px;' line with the text: "Aqui foi aplicado uma margin de 50 px para os 4 lados - Top, Right, Bottom e Left (No sentido horário)".

Vamos como o exemplo usado no dia a dia ficou renderizado no navegador conforme imagem abaixo:

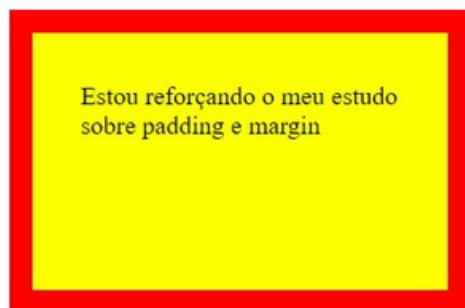


# Exemplos Padding e Margin

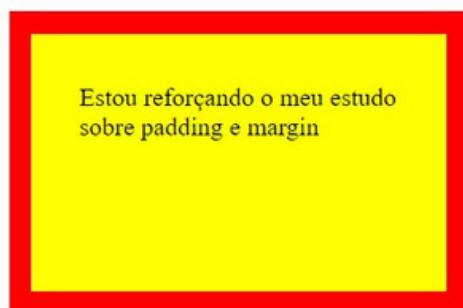
← → C ⓘ 127.0.0.1:5500/index.html



Agora vamos ver abaixo um comparativo lado a lado tanto da forma didática quanto da forma usada no dia a dia depois que ambas foram compiladas no navegador



**Forma  
Didática**



**Forma  
Usada dia a dia**



/igor-rebolla

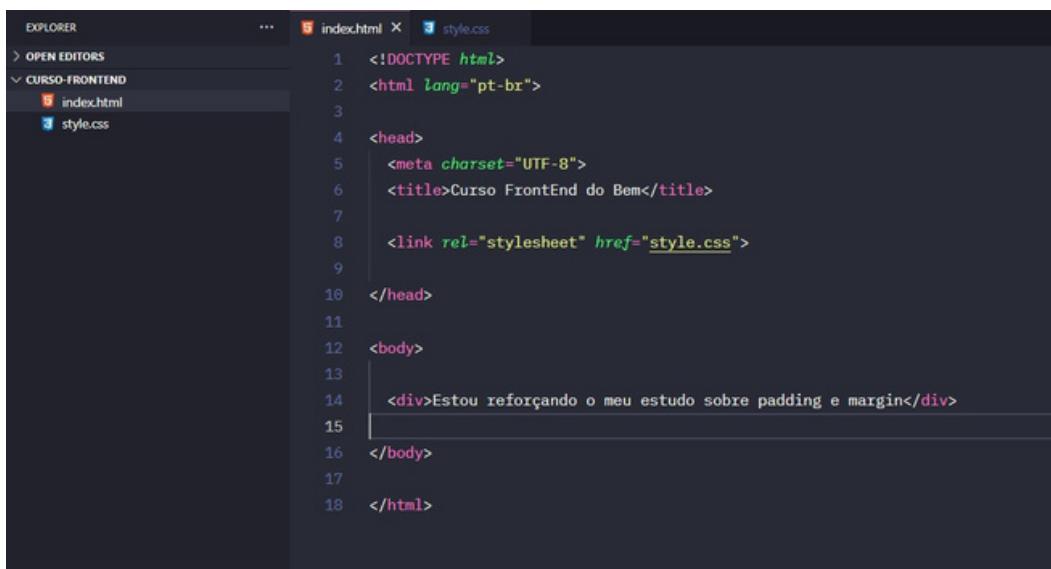
# Exemplos Padding e Margin

Vamos aproveitar a estrutura do arquivo index.html da aula 07 conforme imagem abaixo (Exemplo 2 - Forma didática):

Obs.: Notem que tanto no exemplo 1 quanto no exemplo 2 o qual veremos agora o arquivo index.html não sofreu nenhuma alteração, o que vamos trabalhar é o arquivo style.css

Forma didática:

padding: 30px 15px 30px 15px;  
margin: 40px 25px 40px 25px;



```
EXPLORER          ...   index.html x  style.css
OPEN EDITORS
CURSO-FRONTEND
index.html
style.css

1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>Curso FrontEnd do Bem</title>
7
8    <link rel="stylesheet" href="style.css">
9
10 </head>
11
12 <body>
13
14   <div>Estou reforçando o meu estudo sobre padding e margin</div>
15
16 </body>
17
18 </html>
```

Agora vamos ver na imagem da próxima página como a (forma didática) do Exemplo 2 foi digitada no nosso arquivo style.css:



# Exemplos Padding e Margin



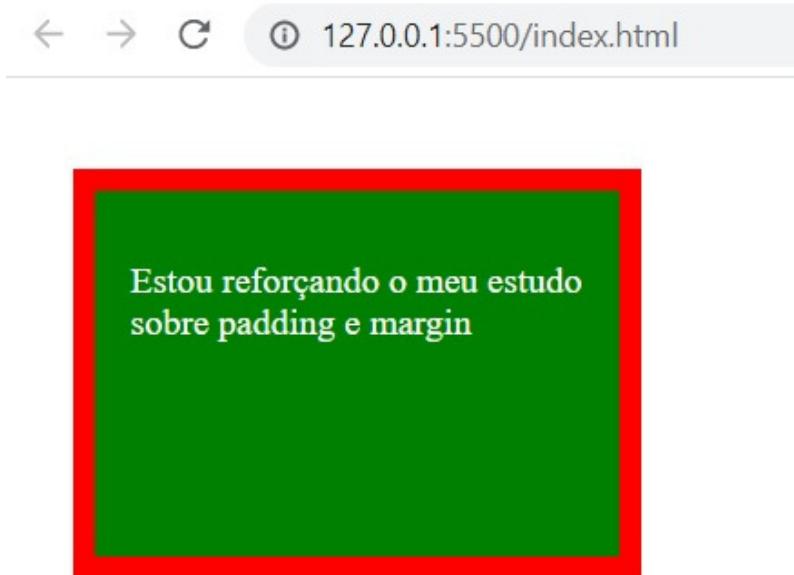
A screenshot of a code editor showing a file named style.css. The code defines a single div element with the following properties:

```
div {  
    background: green;  
    color: white;  
    width: 200px;  
    height: 100px;  
    padding: 30px 15px 30px 15px;  
    border: 10px solid red;  
    margin: 40px 25px 40px 25px;  
}
```

Two annotations are present:

- A red arrow points to the padding declaration with the text: "Aqui conforme Exemplo 2 (didático) por ser algo mais específico nós definimos: padding-top: 30px, padding-right: 15px, padding-bottom: 30px, padding-left: 15px. Sempre lembrando da leitura feita no sentido hóriario começando no padding-top e terminando no padding left."
- A red box highlights the margin declaration with the text: "Aqui conforme Exemplo 2 (didático) por ser algo mais específico nós definimos: margin-top: 30px, margin-right: 15px, margin-bottom: 30px, margin-left: 15px. Sempre lembrando da leitura feita no sentido hóriario começando na margin-top e terminando na margin-left."

Vamos como o exemplo 2 (didático) ficou renderizado no navegador conforme imagem abaixo:



/igor-rebolla

# Exemplos Padding e Margin

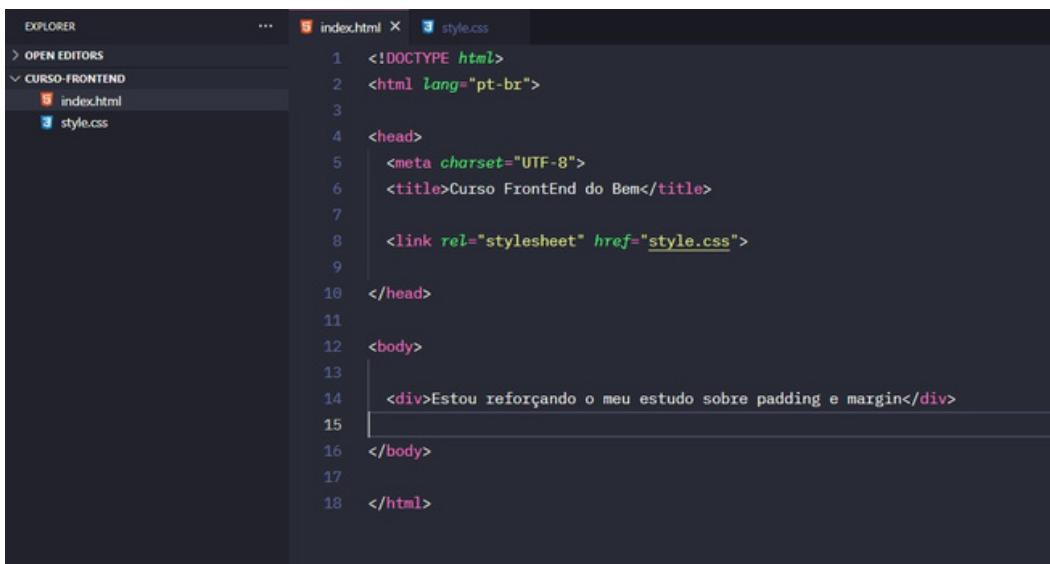
Vamos aproveitar a estrutura do arquivo index.html da aula 07 conforme imagem abaixo (Exemplo 2 – Forma Usada no dia a dia):

Obs.: Notem que tanto no exemplo 1 quanto no exemplo 2 o qual veremos agora o arquivo index.html não sofreu nenhuma alteração, o que vamos trabalhar é o arquivo style.css

Forma didática:

padding: 30px 15px;

margin: 40px 25px;



The screenshot shows a code editor interface with two tabs open: 'index.html' and 'style.css'. The 'index.html' tab contains the following HTML code:

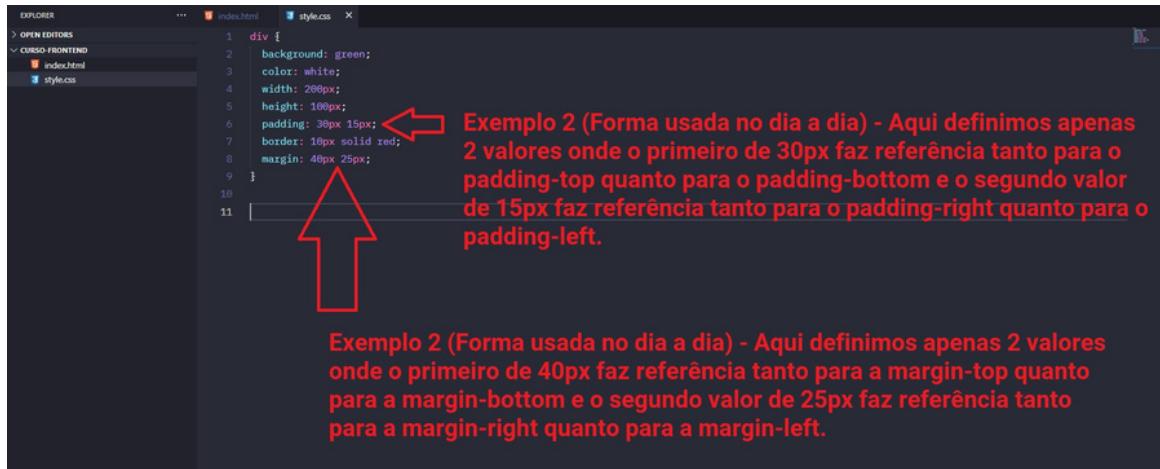
```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Curso FrontEnd do Bem</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div>Estou reforçando o meu estudo sobre padding e margin</div>
</body>
</html>
```

The 'style.css' tab is currently selected but contains no visible code.

Agora vamos ver abaixo como o exemplo 2 (forma usada no dia a dia) foi digitada no nosso arquivo style.css:



# Exemplos Padding e Margin

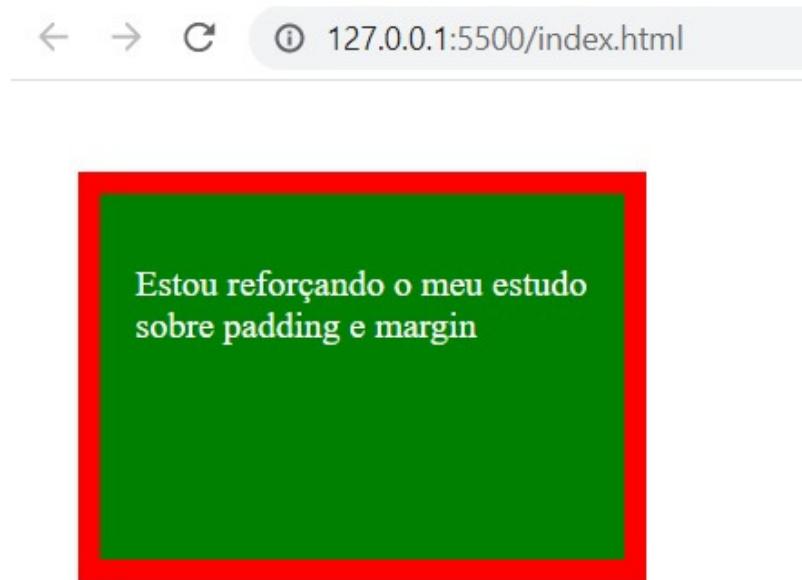


```
1 div {  
2     background: green;  
3     color: white;  
4     width: 200px;  
5     height: 100px;  
6     padding: 30px 15px; ↑  
7     border: 10px solid red;  
8     margin: 40px 25px;  
9 }  
10  
11
```

Exemplo 2 (Forma usada no dia a dia) - Aqui definimos apenas 2 valores onde o primeiro de 30px faz referência tanto para o padding-top quanto para o padding-bottom e o segundo valor de 15px faz referência tanto para o padding-right quanto para o padding-left.

Exemplo 2 (Forma usada no dia a dia) - Aqui definimos apenas 2 valores onde o primeiro de 40px faz referência tanto para a margin-top quanto para a margin-bottom e o segundo valor de 25px faz referência tanto para a margin-right quanto para a margin-left.

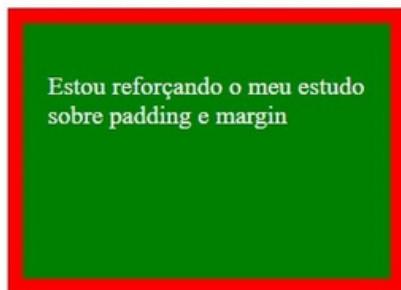
Vamos como o exemplo 2 (didático) ficou renderizado no navegador conforme imagem abaixo:



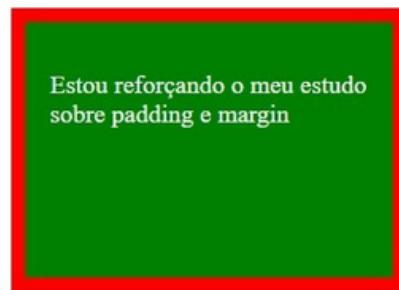
/igor-rebolla

# Exemplos Padding e Margin

Agora vamos ver abaixo um comparativo lado a lado tanto da forma didática quanto da forma usada no dia a dia (Exemplo 2) depois que ambas foram compiladas no navegador:



**Forma  
Didática**



**Forma  
Usada dia a dia**



/igor-rebolla

# Sugestões de prática para essa aula:

1. Abrir o Microsoft Visual Studio Code, depois a pasta Curso-FrontEnd e os arquivos: index.html e style.css;
2. Na pagina 3 desse documento, respondendo a pergunta. Eu disse que não precisaríamos usar de uma só vez o: Top, Right, Bottom e Margin (Tanto para o padding quanto para a margin) - Nós vimos nessa aula tanto a forma didática quanto a forma usada no dia a dia (só que mesmo na forma usual onde definimos apenas 2 valores tanto para o padding quanto para o margin - esses 2 valores (estão representando de forma mais objetiva os 4 valores que digitamos na forma didática. Minhas sugestão é que vocês tentem isolar as propriedades:  
Aproveitem a estrutura do index.html (se quiserem alterar show, pois a prática irá potencializar Vossos aprendizados).
  1. Alterar o padding-top (salvar e visualizar no navegador)
  2. Alterar o padding-right (salvar e visualizar no navegador)
  3. Alterar o padding-bottom (salvar e visualizar no navegador)
  4. Alterar o padding-left (salvar e visualizar no navegador)
  5. Alterar a margin-top (salvar e visualizar no navegador)
  6. Alterar a margin-right (salvar e visualizar no navegador)
  7. Alterar a margin-bottom (salvar e visualizar no navegador)
  8. Alterar a margin-left (salvar e visualizar no navegador)



# O que é FlexBox?

FlexBox é uma solução de layout para conseguirmos alinhar e distribuir itens dentro de um container. Quando o tamanho da página é desconhecido ou quando ele é dinâmico e isso reflete na forma como desenvolvemos aplicações para web hoje em dia.

Porque se fomos parar pra pensar (um site hoje em dia) ele é acessado não só do desktop, mas também do smartphone, do tablet e por ai vai. Todos esses dispositivos eles tem resoluções e tamanhos de tela diferente ou seja: quando vamos desenvolver um site temos que pensar que a pessoa que vai acessar pode ter um smartphone com a tela pequeninha ou até aquelas Tv's de 85 polegadas.

Então quando vamos criar um site, precisamos pensar no layout dele de uma forma que ele seja facilmente adaptado (e esse conceito tem um nome, um não sei se eu falo agora ou deixo para as próximas aulas, esse conceito de adaptar o layout do site para tamanhos de telas diferentes é chamado de Responsivo) ops falei!!! A ideia é deixar esse conceito de Responsivo plantado na cabeça de vocês (vamos ver esse conceito em uma aula mais pra frente um pouquinho desse curso).

Resumindo:

O FlexBox cria containers e itens dentro desses containers flexíveis que se ajustam a essas diferentes resoluções de tela.



# O que é FlexBox?

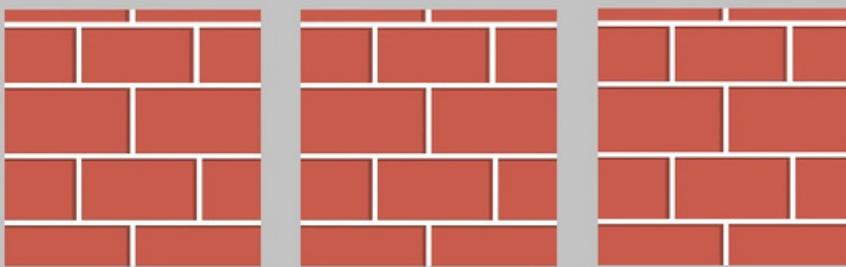
A partir de agora vamos entender sobre as propriedades do FlexBox:

- Aquelas que usamos nos containers;
- Aquelas que usamos dentro dos itens que estão dentro desses containers

Se fizermos aquela analogia marota para a nossa casa, ficaria assim:

- A parede seria o container;
- E os revestimentos (tijolinho) os itens dentro desse container(parede).

**Essa parede cinza virtualmente rebocada é o nosso CONTAINER**



**E cada revestimento(tijolinho) desses representam um item dentro do CONTAINER parede.**

Vamos entender isso melhor com os conceitos dessas propriedades e exemplos já na próxima página dessa aula.

Partiu.....

# Propriedades do FlexBox

Para começarmos a utilizar o Flexbox vamos precisar de um container e o elemento que vamos usar será uma `<div> </div>` para representar esse container. E nele vamos utilizar a propriedade `display` como `flex` para assim criarmos um container que seja flexível. Conforme estrutura abaixo:

```
.container {  
    display: flex;  
}
```

Ou `display: inline-flex` (para criarmos um container que seja flexível a nível de linha). Conforme estrutura abaixo:

```
.container {  
    display: inline-flex;  
}
```



# Propriedades do FlexBox

## FLEX-DIRECTION:

A partir disso podemos utilizar a propriedade flex-direction para organizar os itens dentro do flex container. Temos 4 valores para essa propriedade:

- Row: organiza os itens em linha da esquerda para a direita:



A estrutura base do código que representa o **row** (o qual digitaremos dentro do arquivo **style.css**) será essa abaixo:

```
.container {  
  display: flex;  
  flex-direction: row;  
}
```

**OBS.: Por padrão quando definimos o display como flex ele automaticamente entende que o flex-direction é row (então nesse caso específico não precisaria ser colocado o flex-direction: row).**



# Propriedades do FlexBox

- Row-Reverse: organiza os itens em linha da direita para a esquerda:



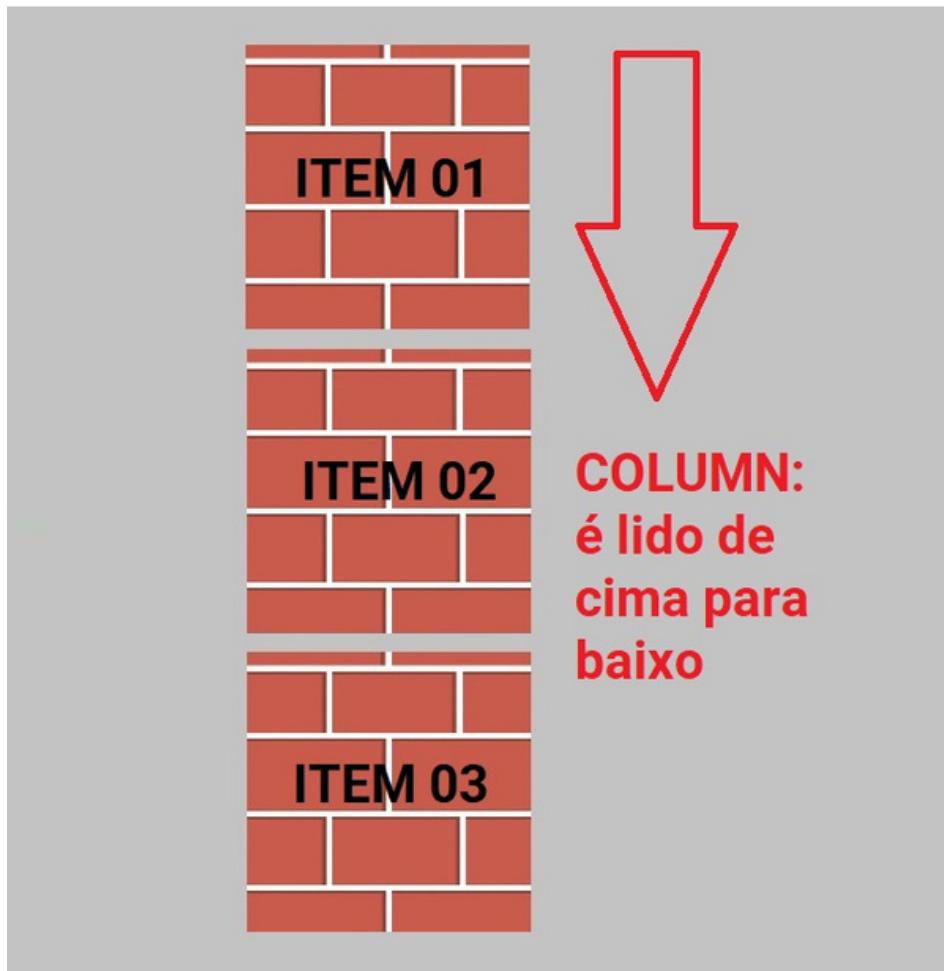
A estrutura base do código que representa o **row-reverse** (o qual digitaremos dentro do arquivo **style.css**) será essa abaixo:

```
.container {  
  display: flex;  
  flex-direction: row-reverse;  
}
```



# Propriedades do FlexBox

- Column: organiza os itens em coluna de cima para baixo



A estrutura base do código que representa o **column** (o qual digitaremos dentro do arquivo **style.css**) será essa abaixo:

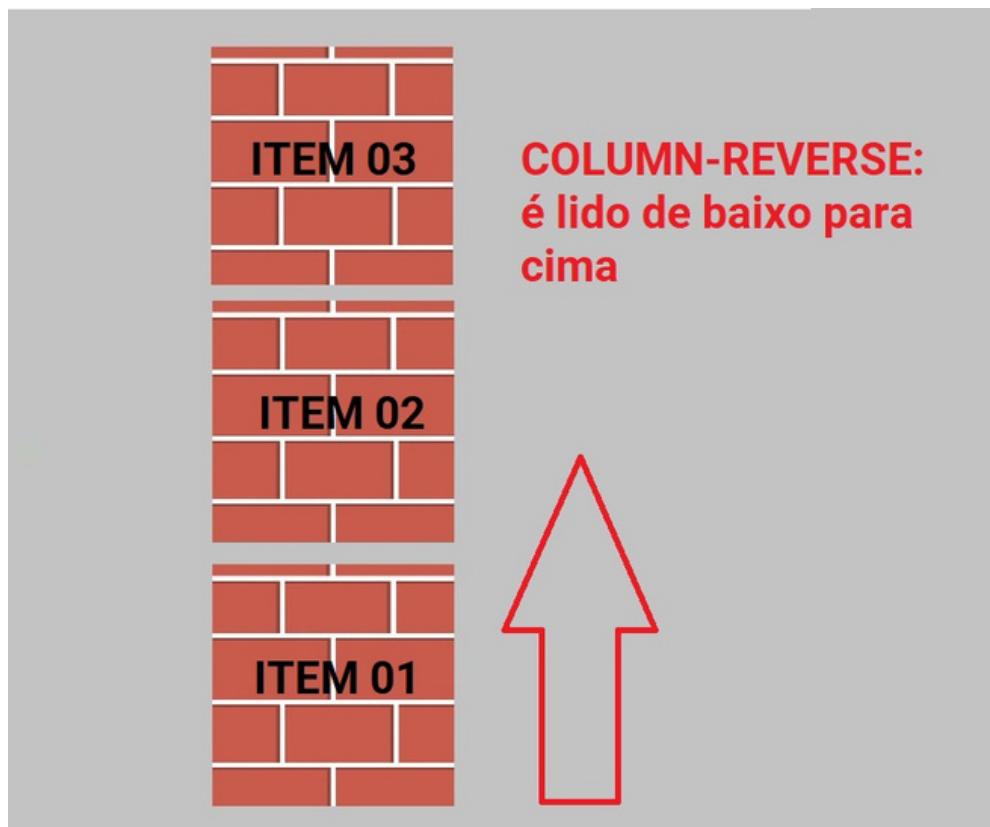
```
.container {  
  display: flex;  
  flex-direction: column;  
}
```



/igor-rebolla

# Propriedades do FlexBox

- Column-Reverse: organiza os itens em coluna de baixo para cima:



A estrutura base do código que representa o **column-reverse** (o qual digitaremos dentro do arquivo **style.css**) será essa abaixo:

```
.container {  
display: flex;  
flex-direction: column-reverse;  
}
```



/igor-rebolla

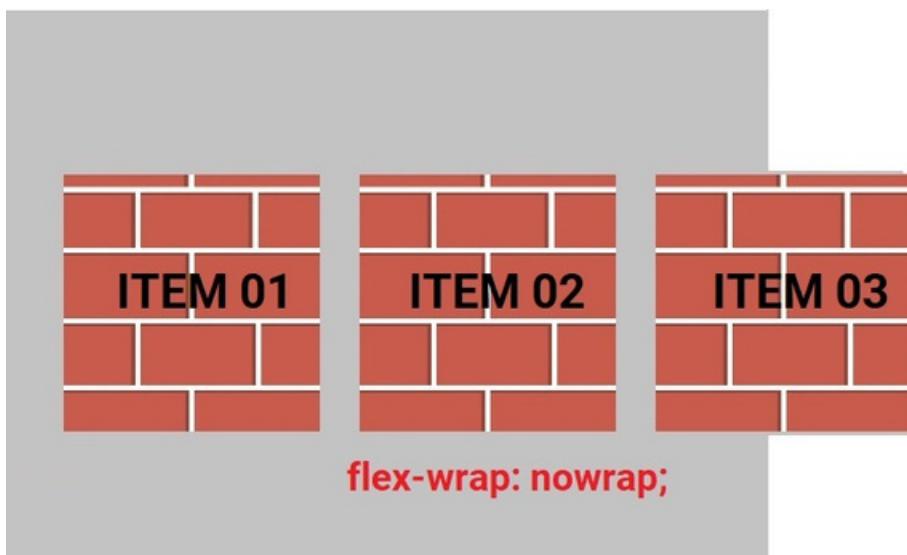
# Propriedades do FlexBox

## FLEX-WRAP:

No FlexBox a largura de um container ou de um item é ajustada automaticamente de acordo com o espaço disponível para eles. E para ajustar esse tipo de comportamento temos a propriedade flex-wrap que possui 3 valores:

- **nowrap** (esse valor já vem como padrão): e não é permitido a quebra de linha.

Notem que no exemplo abaixo temos a valor **nowrap** definido e o que aconteceu: O terceiro item do nosso revestimento(tijolinho) ficou "para fora" do nosso container parede, pois a largura dos (3 flex itens) são maiores do que o tamanho do nosso container parede e como o nowrap não permite a quebra de linha, ele "ajustou" dessa forma:

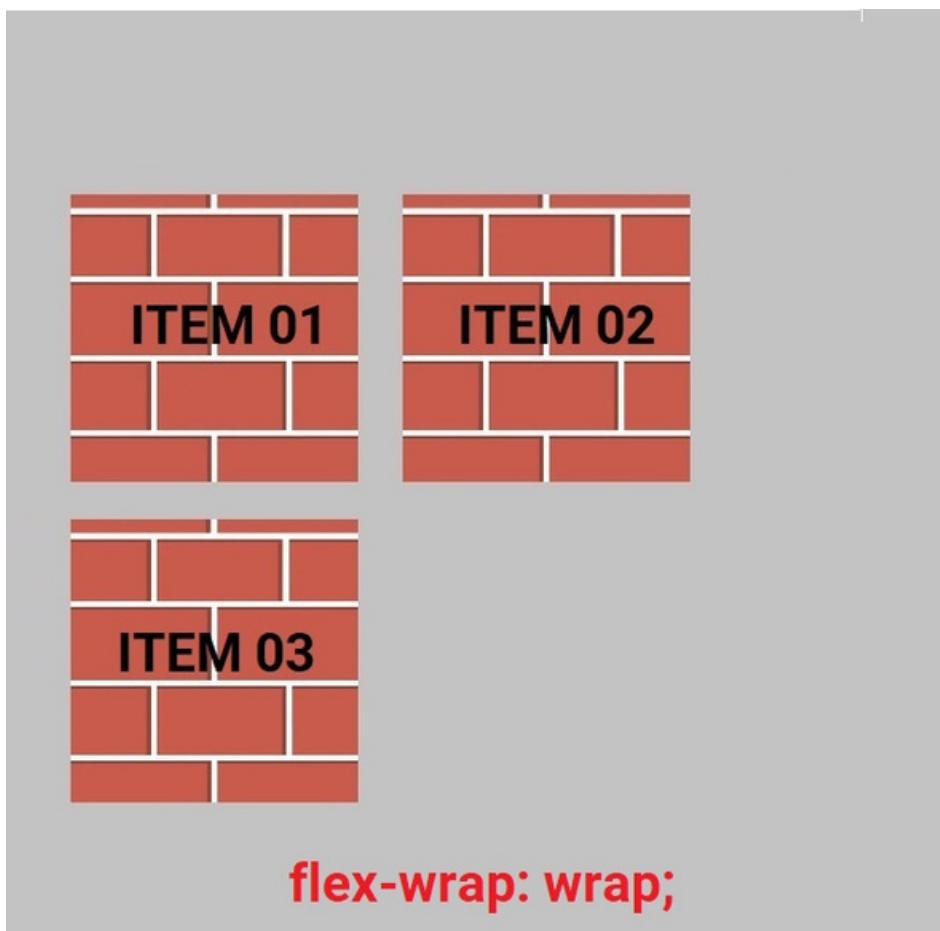


# Propriedades do FlexBox

## FLEX-WRAP:

- **wrap**: aqui a linha é quebrada e os itens que estão na parte mais a direita vão ser gentilmente convidados (leia-se: deslocados) para a linha abaixo.

Notem que no exemplo abaixo temos a valor wrap definido e o que aconteceu: O terceiro item do nosso revestimento(tijolinho) que está mais a direita do nosso container passou para linha de baixo pois usamos a propriedade wrap. E ele ficou ajustado assim:



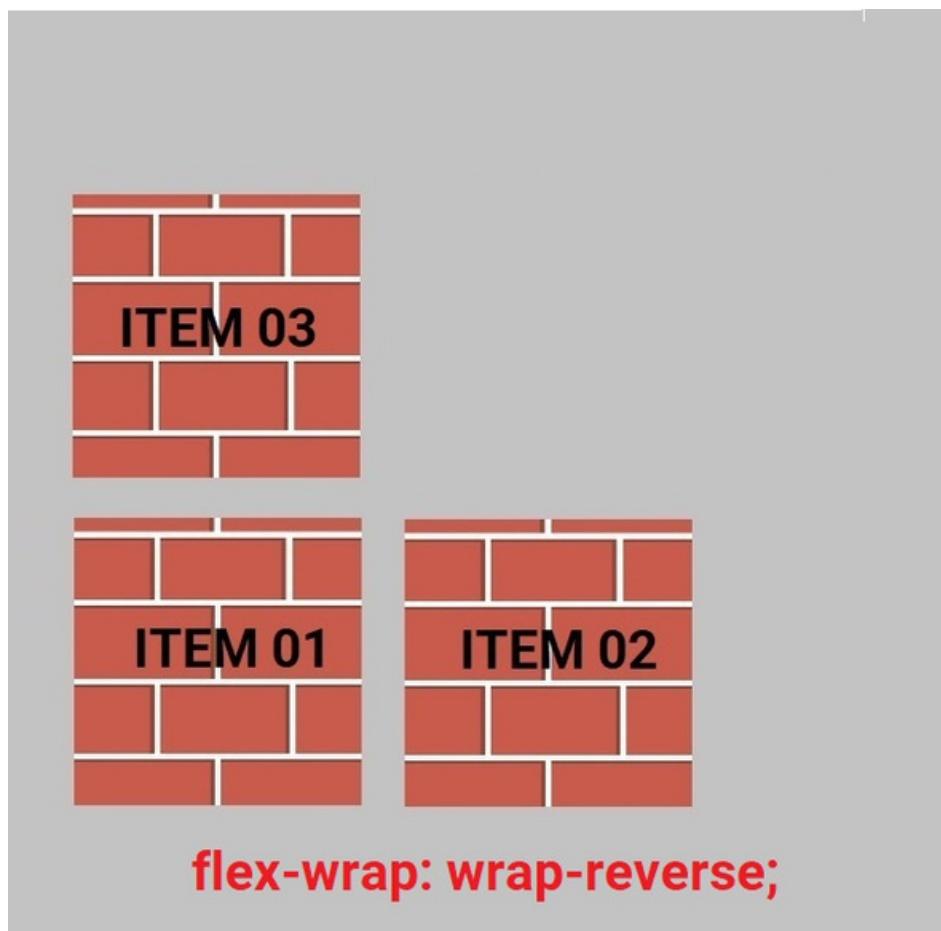
/igor-rebolla

# Propriedades do FlexBox

## FLEX-WRAP:

- **wrap-reverse**: aqui a linha é quebrada e os itens que estão na parte mais a direita vão ser gentilmente convidados (leia-se: deslocados) para a linha acima

Notem que no exemplo abaixo temos a valor **wrap-reverse** definido e o que aconteceu: O terceiro item do nosso revestimento(tijolinho) que está mais a direita do nosso container passou para linha acima (de forma reversa) pois usamos a propriedade wrap-reverse. E ele ficou ajustado assim:



/igor-rebolla

# Propriedades do FlexBox

## JUSTIFY-CONTENT:

- **justify-content:** vai definir o alinhamento dos itens em um container em relação ao eixo-principal. O justify-content ajuda a distribuir o espaço entre esses itens principalmente quando eles já atingiram o tamanho máximo.

A propriedade justify-content possui **5 valores** que poderão ser usados de acordo com a necessidade. Essas propriedades são:

### **justify-content: flex-start;**

Aqui os itens são alinhados no inicio

### **justify-content: flex-end;**

Aqui os itens são alinhados no final

### **justify-content: center;**

Aqui os itens são alinhados ao centro

### **justify-content: space-between;**

Aqui o primeiro item tem o seu deslocamento no inicio, enquanto o último item tem o seu deslocamento para o final e os itens restantes são distribuidos igualmente entre eles.

### **justify-content: space-around;**

Aqui os itens são distribuidos igualmente entre eles. Com a diferença que o primeiro e o último item não vão ficar grudados nas bordas do container.

Vamos ver nas próximas páginas (as estruturas de cada uma das 5 propriedades do justify-content e um exemplo para melhor entendimento).



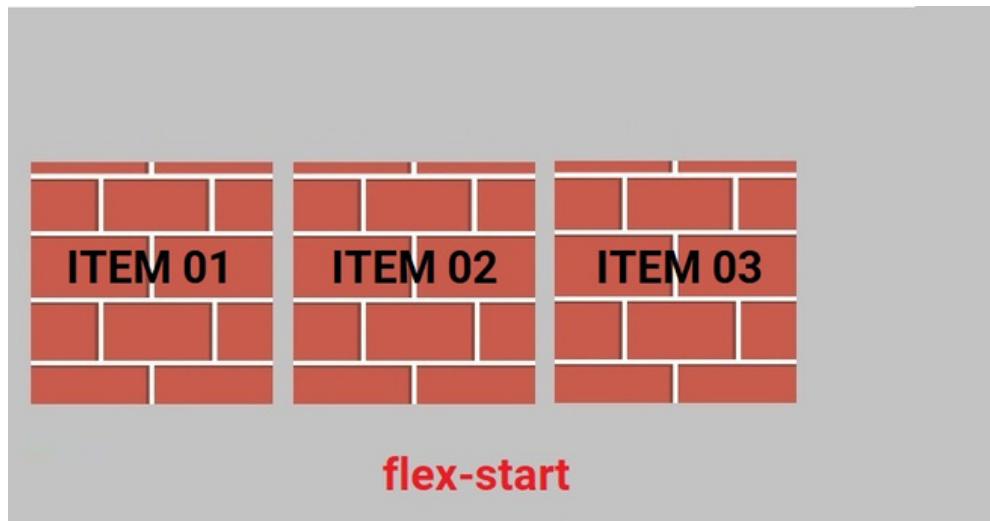
# Propriedades do FlexBox

**justify-content: flex-start;**

Aqui os itens são alinhados no inicio

Estrutura básica:

```
. container {  
    display: flex;  
    justify-content: flex-start;
```



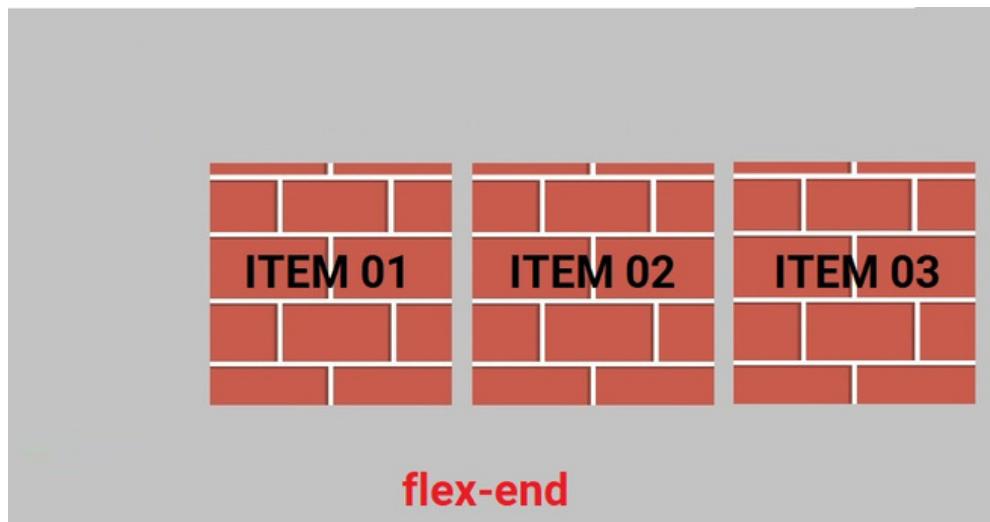
# Propriedades do FlexBox

**justify-content: flex-end;**

Aqui os itens são alinhados no final

Estrutura básica:

```
. container {  
  display: flex;  
  justify-content: flex-end;  
}
```



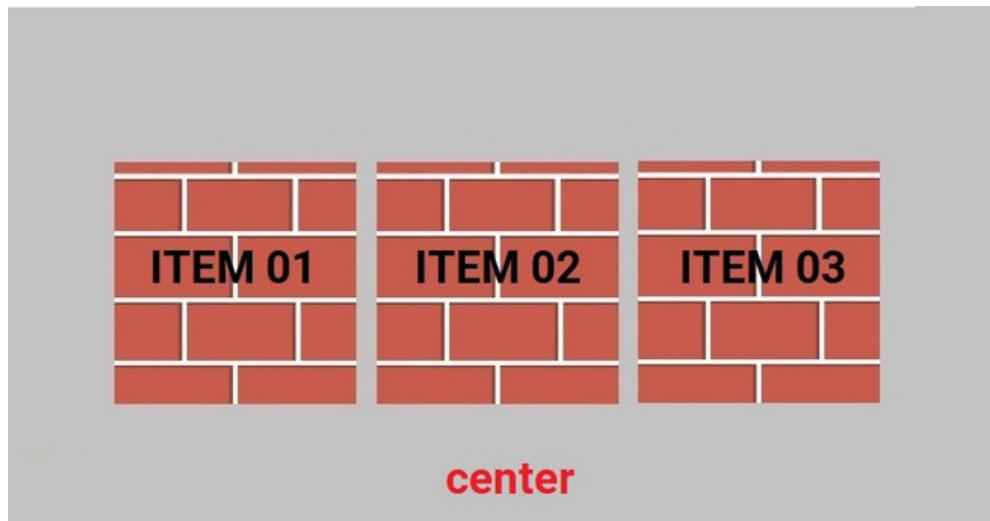
# Propriedades do FlexBox

**justify-content: center;**

Aqui os itens são alinhados ao centro

Estrutura básica:

```
.container {  
  display: flex;  
  justify-content: center;  
}
```



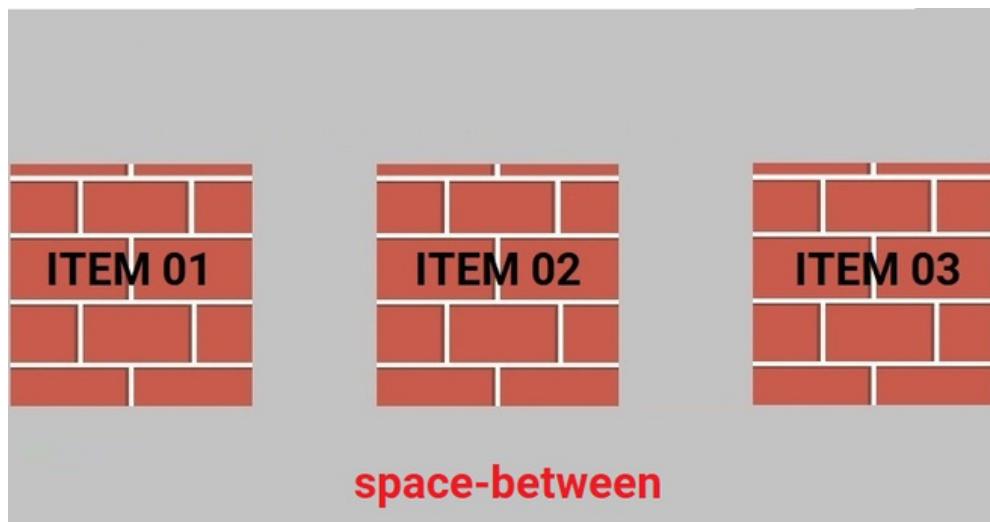
# Propriedades do FlexBox

## **justify-content: space-between;**

Aqui o primeiro item tem o seu deslocamento no inicio, enquanto o último item tem o seu deslocamento para o final e os itens restantes são distribuidos igualmente entre eles. Ops, quase me esqueci (o item mais a esquerda e o item mais a direita ficam grudados em suas respectivas bordas).

Estrutura básica:

```
. container {  
display: flex;  
justify-content: space-between;  
}
```



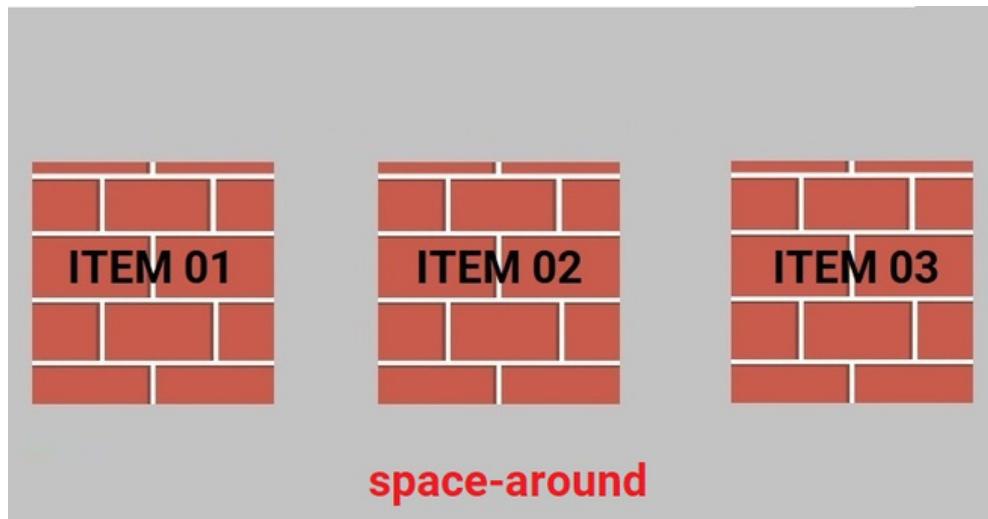
# Propriedades do FlexBox

## **justify-content: space-around;**

Aqui os itens são distribuídos igualmente entre eles. Com a diferença que o primeiro e o último item não vão ficar grudados nas bordas do container.

Estrutura básica:

```
.container {  
  display: flex;  
  justify-content: around;  
}
```



# Exemplos do FlexBox

Vamos ver agora exemplos na prática dentro dos arquivos:

**index.html** (estrutura);

**style.css** (local onde vamos digitar as propriedades e os valores do FlexBox).

O código no index.html para exemplo terá essa estrutura, conforme imagem abaixo:

- 1 div (onde foi adicionada a class = container-parede)
- Outras 3 div's com a class= item-revestimento que estão dentro dessa class = container-parede.

```
EXPLORER      ...
OPEN EDITORS
  CURSO-FRONTEND
    indexhtml
    style.css

1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>Aula 09 - FlexBox</title>
7
8    <link rel="stylesheet" href="style.css">
9
10 </head>
11
12 <body>
13
14   <div class="container-parede">
15     <div class="item-revestimento">Item 01</div>
16     <div class="item-revestimento">Item 02</div>
17     <div class="item-revestimento">Item 03</div>
18   </div>
19
20 </body>
21
22 </html>
```



/igor-rebolla

# Exemplos do FlexBox

Aqui a estrutura do código no arquivo style.css:

Na classe: container-parede (aqui no style.css representada pelo .container-parede), é composta por:

```
max-width: 500px  
(definido uma largura máxima do nosso container-parede em 500px  
margin: 0 auto;  
(deixar centralizada quando renderizada no navegador)  
border: 1px solid #black;  
(definido uma borda de 1px, linha sólida na cor preta  
background: grey;  
(cor cinza para simular a cor de uma parede rebocada)
```

Na classe: item-revestimento (aqui no style.css representada pelo .item-revestimento) é composta por:

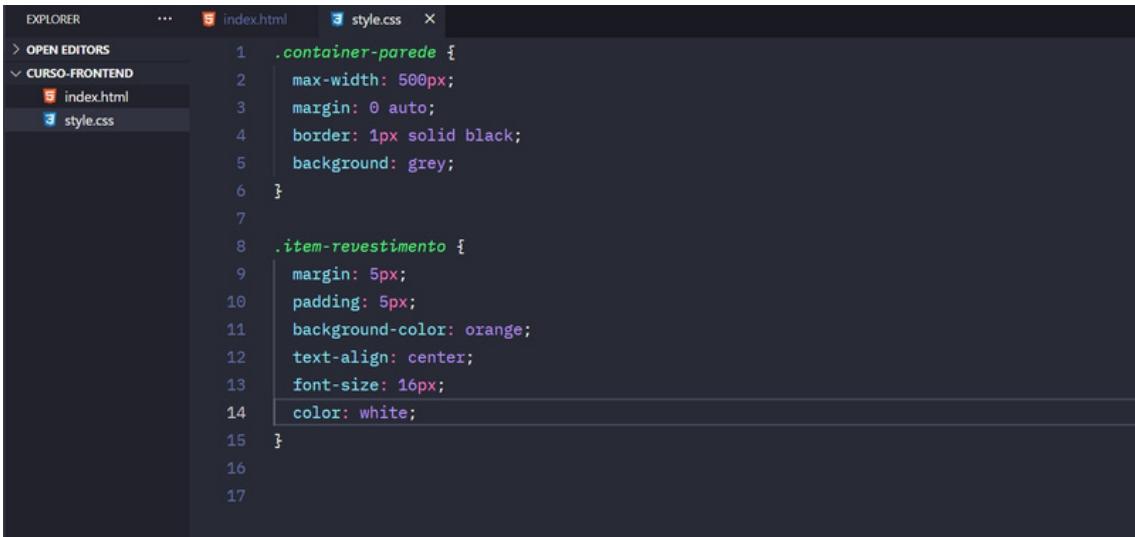
```
margin: 5px;  
(para distanciar um item-revestimento dos outros)  
padding: 5px;  
(definimos um espaçamento interno)  
background-color: orange;  
(definimos a cor de fundo em laranja para simular o revestimento tijolinho)  
text-align: center;  
(texto alinhado no centro)  
font-size: 16px;  
(tamanho da fonte definido em 16px)  
color: white;  
(cor da fonte na cor branca)
```



/igor-rebolla

# Exemplos do FlexBox

O código no style.css inicialmente terá essa estrutura, conforme imagem abaixo:

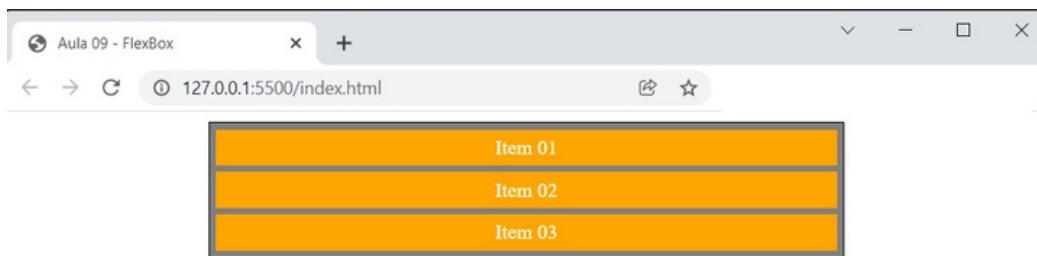


```
EXPLORER ... index.html style.css

OPEN EDITORS
CURSO-FRONTEND
index.html
style.css

1 .container-parede {
2   max-width: 500px;
3   margin: 0 auto;
4   border: 1px solid black;
5   background: grey;
6 }
7
8 .item-revestimento {
9   margin: 5px;
10  padding: 5px;
11  background-color: orange;
12  text-align: center;
13  font-size: 16px;
14  color: white;
15 }
16
17
```

Na imagem abaixo podemos ver o resultado desse código renderizado no navegador

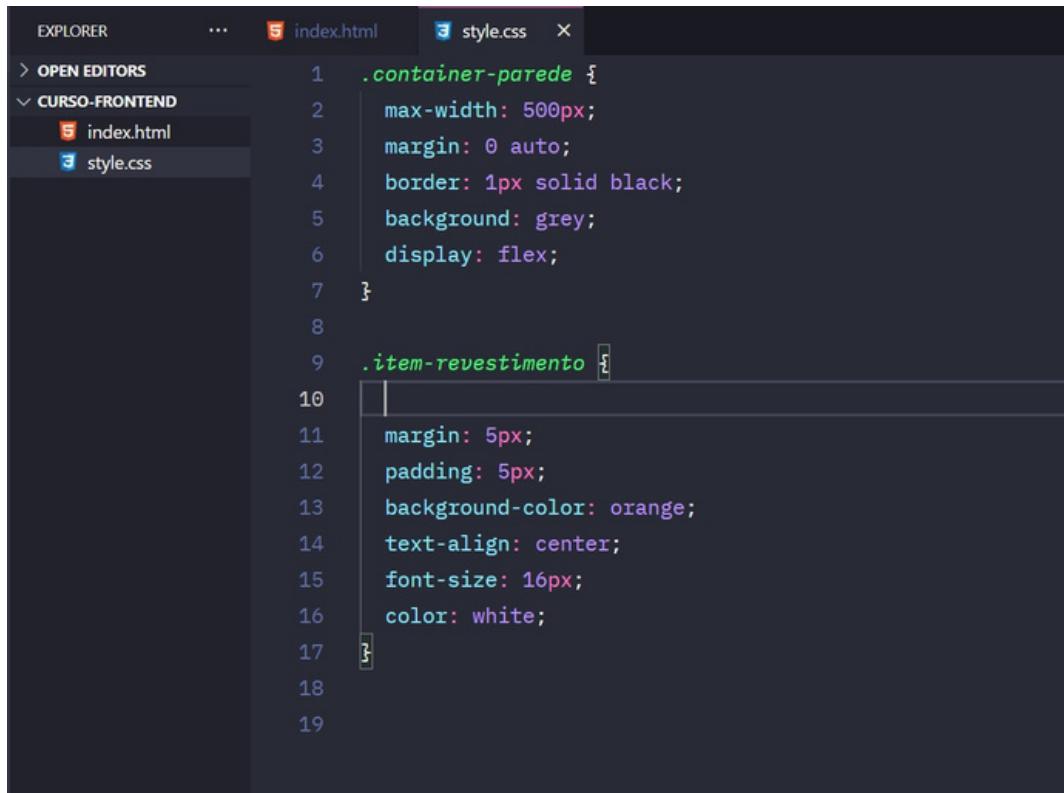


Notem que não colocamos nenhuma propriedade do FlexBox vista nessa aula até o momento. Vamos fazer isso a partir da próxima página.



# Exemplos do FlexBox

Para começarmos a usar o **FlexBox**, precisamos definir antes de tudo o nosso **container-parede** como **display:flex;**



```
EXPLORER      ...  index.html  style.css
> OPEN EDITORS
  CURSO-FRONTEND
    index.html
    style.css

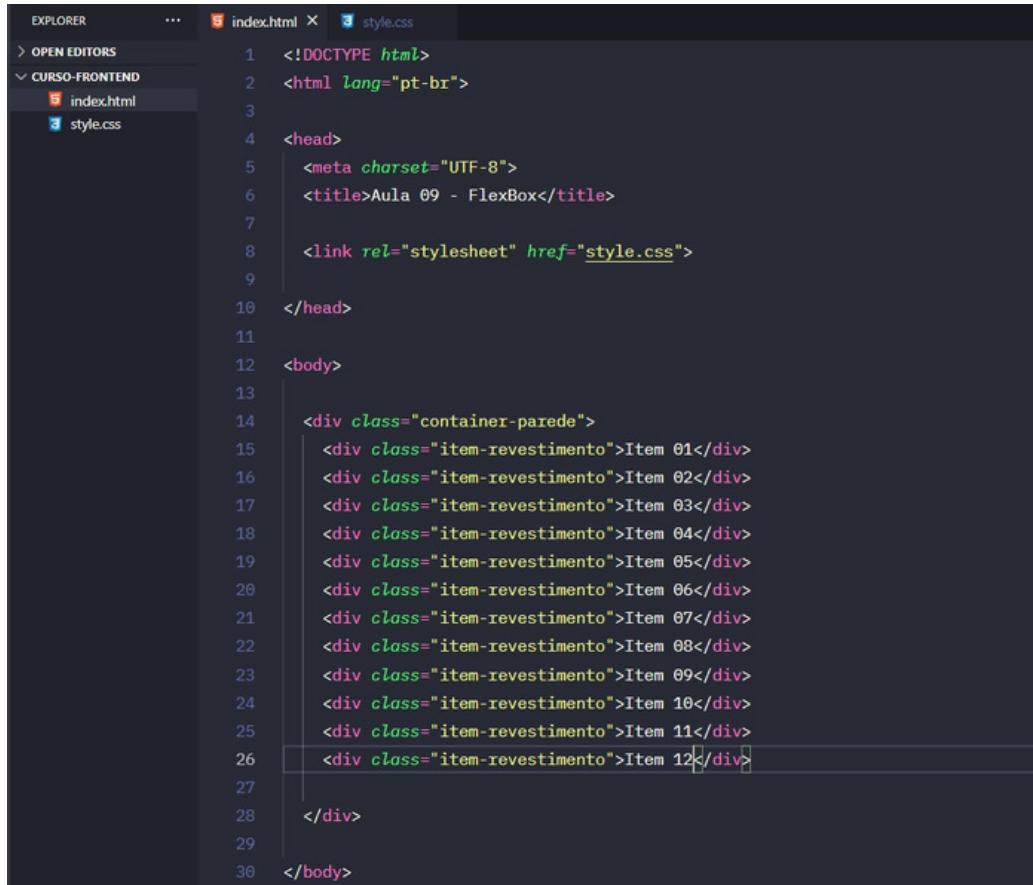
1 .container-parede {
2   max-width: 500px;
3   margin: 0 auto;
4   border: 1px solid black;
5   background: grey;
6   display: flex;
7 }
8
9 .item-revestimento {
10
11   margin: 5px;
12   padding: 5px;
13   background-color: orange;
14   text-align: center;
15   font-size: 16px;
16   color: white;
17
18
19
```

Abaixo vamos ver o código acima renderizado no navegador depois que colocamos o `display:flex;` (os itens que antes estavam em bloco por padrão agora estão em linha):



# Exemplos do FlexBox

Para vermos na prática o funcionamento do nowrap (vamos acrescentar mais 9 div's no nosso arquivo index.html)



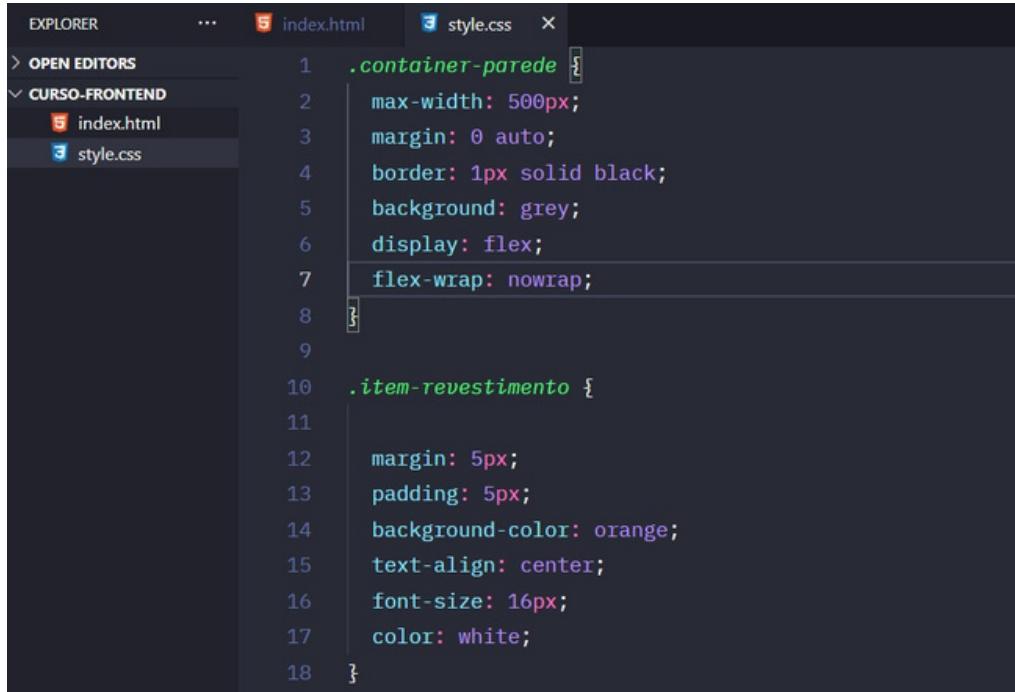
The screenshot shows a code editor interface with two tabs: 'index.html' and 'style.css'. The 'index.html' tab is active, displaying the following HTML code:

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>Aula 09 - FlexBox</title>
7
8    <link rel="stylesheet" href="style.css">
9
10 </head>
11
12 <body>
13
14  <div class="container-parede">
15    <div class="item-revestimento">Item 01</div>
16    <div class="item-revestimento">Item 02</div>
17    <div class="item-revestimento">Item 03</div>
18    <div class="item-revestimento">Item 04</div>
19    <div class="item-revestimento">Item 05</div>
20    <div class="item-revestimento">Item 06</div>
21    <div class="item-revestimento">Item 07</div>
22    <div class="item-revestimento">Item 08</div>
23    <div class="item-revestimento">Item 09</div>
24    <div class="item-revestimento">Item 10</div>
25    <div class="item-revestimento">Item 11</div>
26    <div class="item-revestimento">Item 12</div>
27
28  </div>
29
30 </body>
```

Agora vamos acrescentar no arquivo style.css a propriedade flex-wrap: nowrap; (conforme imagem da próxima página).



# Exemplos do FlexBox

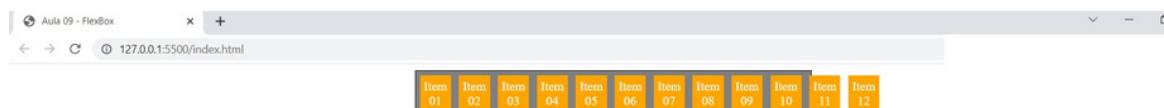


The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows the project structure: **OPEN EDITORS** (empty), **CURSO-FRONTEND** (index.html, style.css).
- index.html**: Preview tab.
- style.css**: Preview tab.
- Code Editor**: Contains the following CSS code:

```
.container-parede {  
    max-width: 500px;  
    margin: 0 auto;  
    border: 1px solid black;  
    background: grey;  
    display: flex;  
    flex-wrap: nowrap;  
}  
  
.item-revestimento {  
    margin: 5px;  
    padding: 5px;  
    background-color: orange;  
    text-align: center;  
    font-size: 16px;  
    color: white;  
}
```

Vamos ver o resultado do código acima renderizado no navegador (conforme imagem abaixo).



Qual o motivo do item 11 e 12 terem saído do nosso container-parede? Por causa do nowrap (esse valor não permite quebra de linha) e ele continua como "senão" houvesse o amanhã sempre em frente caso tivéssemos acrescentado mais 3 div's por exemplo. Para resolver esse "problema" podemos usar wrap (o qual veremos na próxima página).

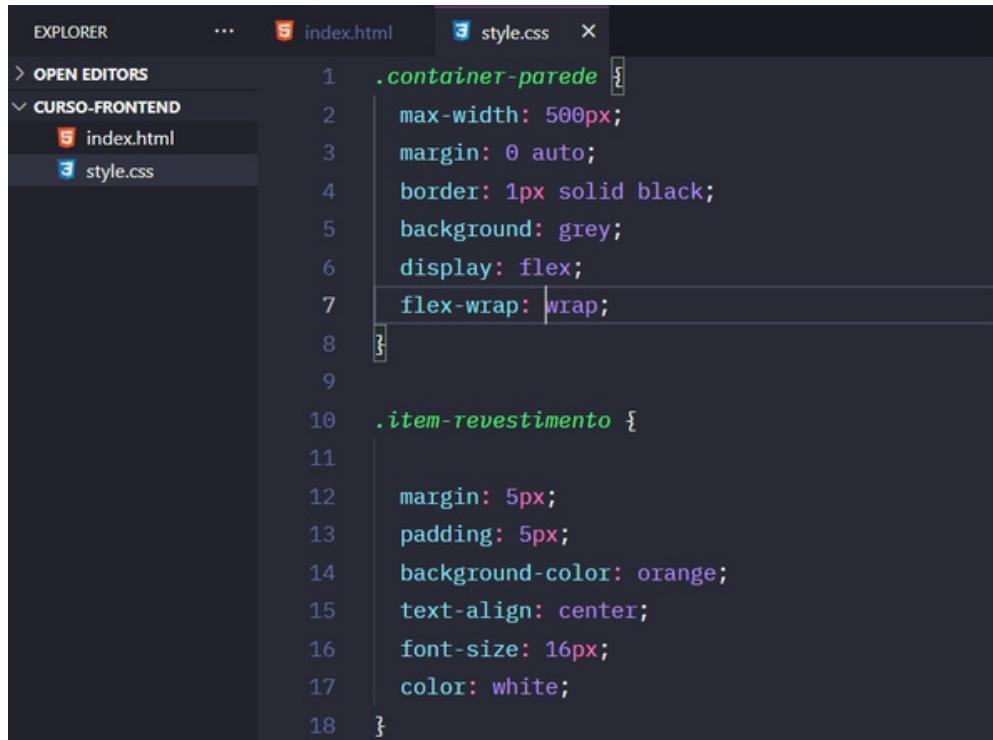
Partiu ver o wrap...



# Exemplos do FlexBox

Agora vamos acrescentar no arquivo style.css a propriedade flex-wrap: wrap; (conforme imagem abaixo).

Ah e vamos manter as 12 div's no arquivo index.html.



```
EXPLORER      ...  index.html  style.css X
OPEN EDITORS
CURSO-FRONTEND
index.html
style.css

1 .container-parede {
2   max-width: 500px;
3   margin: 0 auto;
4   border: 1px solid black;
5   background: grey;
6   display: flex;
7   flex-wrap: wrap;
8 }

9
10.item-revestimento {
11
12  margin: 5px;
13  padding: 5px;
14  background-color: orange;
15  text-align: center;
16  font-size: 16px;
17  color: white;
18 }
```

Vamos ver o resultado do código acima renderizado no navegador (conforme imagem abaixo).



O item 07 foi o último item que cabia no container sendo assim o item 08 foi gentilmente convidado a descer (pois ele não cabia mais na linha de cima) e automaticamente levou o item 09 até o 12 com ele.

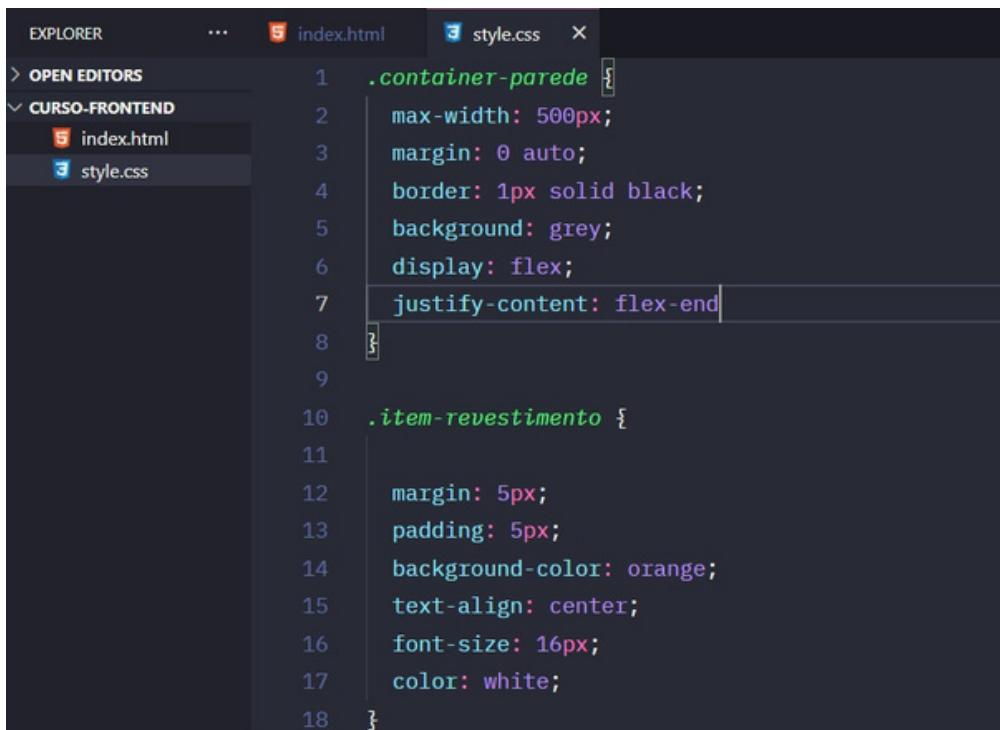


# Exemplos do FlexBox

Agora vamos ver um exemplo de uma das 5 propriedades do justify-content que vimos nessa aula. Bora ver o justify-content: flex-end;

Ah e vamos voltar com as nossas 3 div's no arquivo index.html.

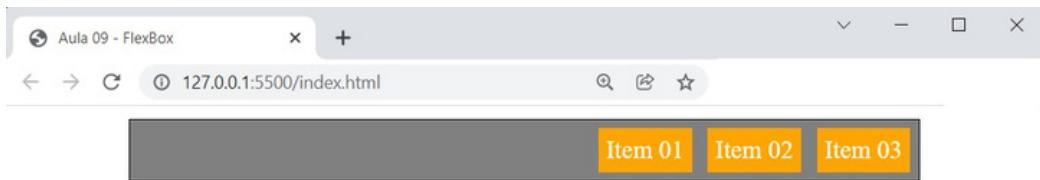
Depois que colocarmos o justify-content: flex-end; (dentro do nosso arquivo style.css) ele vai ficar conforme imagem abaixo:



```
EXPLORER      ...  index.html  style.css X
OPEN EDITORS
CURSO-FRONTEND
index.html
style.css

1 .container-parede {
2   max-width: 500px;
3   margin: 0 auto;
4   border: 1px solid black;
5   background: grey;
6   display: flex;
7   justify-content: flex-end;
8 }
9
10 .item-revestimento {
11
12   margin: 5px;
13   padding: 5px;
14   background-color: orange;
15   text-align: center;
16   font-size: 16px;
17   color: white;
18 }
```

Vamos ver o resultado do código acima renderizado no navegador (conforme imagem abaixo).



/igor-rebolla

# Sugestões de prática para essa aula:

1. Abrir o Microsoft Visual Studio Code, depois a pasta Curso-FrontEnd e os arquivos: index.html e style.css;
2. Nas páginas 18 até 25 dessa aula eu fiz alguns exemplos das propriedade explicadas nas páginas anteriores. Minha sugestão aqui é: Fazer os exemplos que ficaram faltando:
  - flex-wrap: wrap-reverse;
  - justify-content: flex-start;
  - justify-content: center;
  - justify-content: space-between;  
(Obs para o space-between: Na explicação desse valor na página 16 dessa aula eu citei que o space-between tem o item mais a esquerda grudado na sua respectiva borda e o item mais a direita a mesma (isso é o padrão da propriedade) nada impede de colocar uma margin de 5px por exemplo e dar respectivos espaçamentos.
  - justify-content: space-around;
  - flex-direction: row;
  - flex-direction: row-reverse;
  - flex-direction: column;
  - flex-direction: column-reverse;
3. Ah e bora fazer os exemplos mostrados nessa aula também.



# O que é CSS Grid?

Assim como o CSS FlexBox visto na aula passada, o CSS Grid também é uma solução de layout que nos permite alinhar e distribuir itens dentro de um container. A principal diferença entre os 2 é que:

O CSS FlexBox é unidimensional (Atuando no Eixo X);

O CSS Grid é bidimensional (Atuando tanto no Eixo X quanto no Eixo Y)

E através dessa junção de linhas e colunas tanto do Eixo X quanto no Eixo Y vai formar literalmente uma grade (por isso chamamos de Grid).

Resumindo:

O CSS Grid cria containers e dentro desses containers são criados itens em forma de grade, o que nos permite:

- definir onde esse itens irão ficar;
- informar qual será o seu tamanho;
- e qual será o valor do espaçamento entre eles

Ahhhhh já ia me esquecendo: Assim como o FlexBox o CSS Grid também se ajusta em diferentes tamanho de tela (Lembram do Responsivo que eu mencionei na aula passada, com o intuito de plantar a sementinha em Vossas cabeças, olha ele aqui novamente).



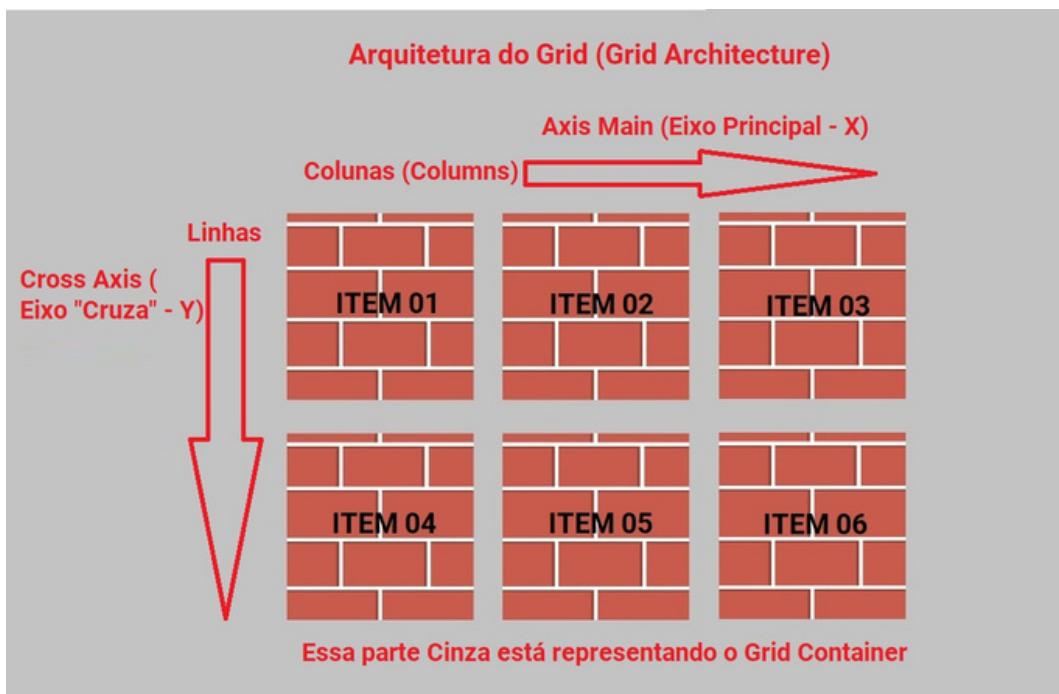
# O que é CSS Grid?

A partir de agora vamos entender sobre as propriedades do CSS Grid:

- Aquelas que usamos nos containers;
- Aquelas que usamos dentro dos itens que estão dentro desses containers

Se fizermos aquela analogia marota para a nossa casa, a Arquitetura ficaria assim:

- A parede (parte cinza) seria o container;
- E os revestimentos (tijolinho) os itens dentro desse grid container(parede).



Vamos entender isso melhor com os conceitos dessas propriedades e exemplos já na próxima página dessa aula.

Partiu...



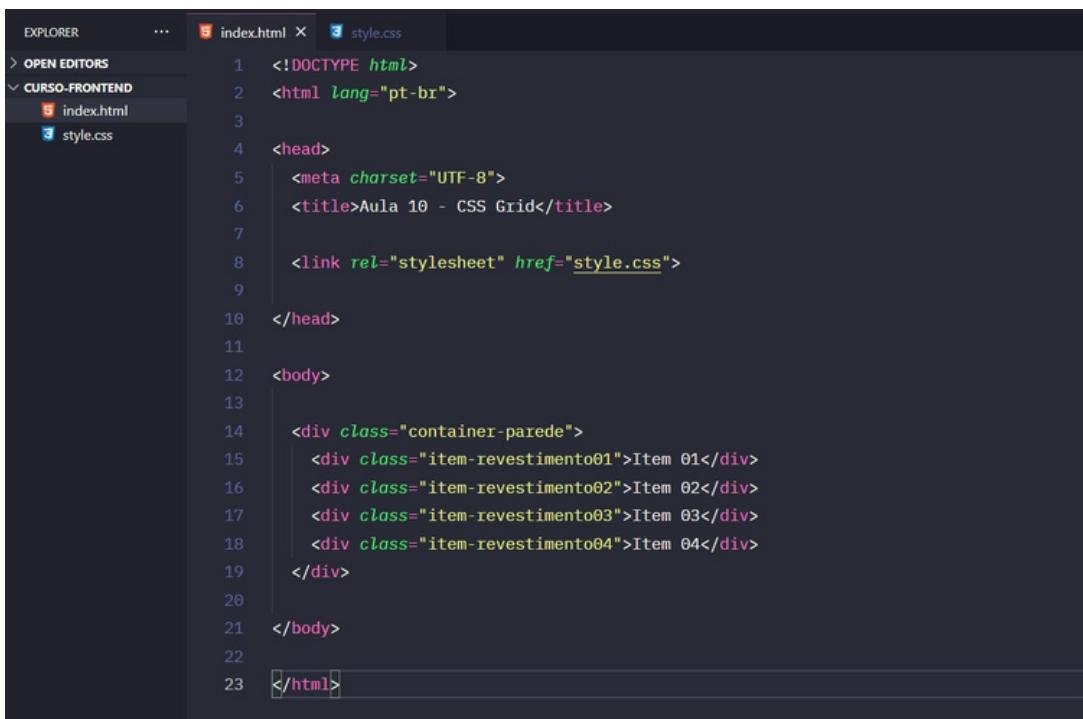
# Propriedades do CSS Grid

Para começarmos a utilizar o CSS Grid vamos precisar de um container e o elemento que vamos usar será uma `<div>` `</div>` para representar esse container. E nele vamos utilizar a propriedade `display` como grid.

Conforme estrutura abaixo:

```
.container {  
    display: grid;  
}
```

Vamos ver abaixo como ficou a estrutura do arquivo `index.html` a qual iremos usar para entender como o código é renderizado no navegador.



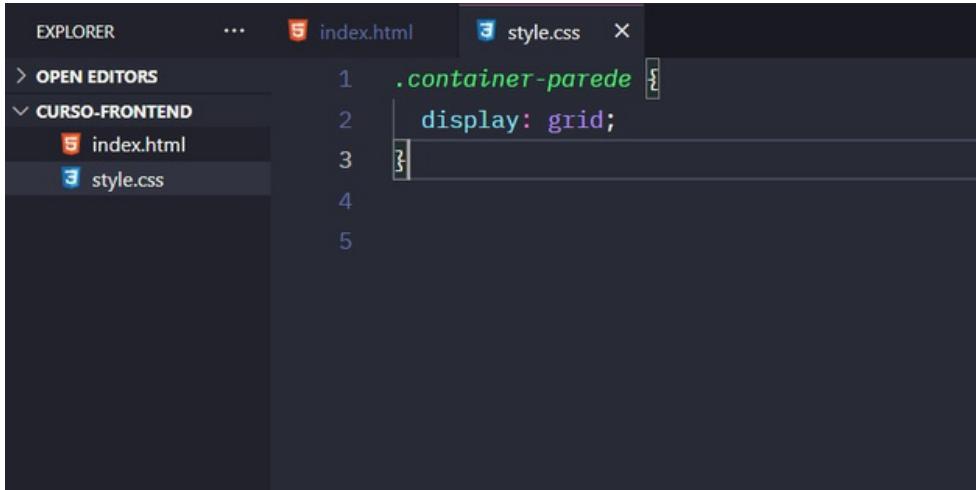
The screenshot shows a code editor interface with two tabs open: `index.html` and `style.css`. The `index.html` tab is active, displaying the following HTML code:

```
1  <!DOCTYPE html>  
2  <html lang="pt-br">  
3  
4      <head>  
5          <meta charset="UTF-8">  
6          <title>Aula 10 - CSS Grid</title>  
7  
8          <link rel="stylesheet" href="style.css">  
9  
10     </head>  
11  
12     <body>  
13  
14         <div class="container-parede">  
15             <div class="item-revestimento01">Item 01</div>  
16             <div class="item-revestimento02">Item 02</div>  
17             <div class="item-revestimento03">Item 03</div>  
18             <div class="item-revestimento04">Item 04</div>  
19         </div>  
20  
21     </body>  
22  
23 </html>
```

Estrutura definida no arquivo `index.html` vamos ver como informamos ao arquivo `style.css` que queremos utilizar o `display: grid`.



# Propriedades do CSS Grid



```
EXPLORER      ...  index.html  style.css  X
> OPEN EDITORS
CURSO-FRONTEND
  index.html
  style.css
```

```
1 .container-parede {
2   display: grid;
3 }
```

E bora ver o resultado disso renderizado no navegador:



Eita, Igor... esse tal de `display: grid` é um "brincalhão" não fez nada? Olhando "de primeira", realmente parece que ficou como se tivéssemos definido um `display` como `block`, já que os itens ficaram alinhados um embaixo do outro e sem nenhuma coluna definida.

Vamos ver o mesmo exemplo, só que agora fazendo analogia com a parede da nossa casa:



# Propriedades do CSS Grid



Assim como no exemplo da página anterior os itens-revestimento (tijolinhos) ao inserirmos o display como grid (ficaram alinhados) um embaixo do outro como se nada tivesse acontecido. Mas nos bastidores ao informarmos para o nosso arquivo style que o display é grid... Sinalizamos mais ou menos assim: "style.css já foi definido o display como grid; agora descansa um pouquinho ai e aguarde o próximo comando para que as coisas comecem a fazer sentido e sejam mostradas na tela.

E a primeira propriedade que iremos ver é o grid-template-columns.



# Propriedades do CSS Grid

## Grid-Template-Columns:

Essa é a propriedade dentro do CSS Grid que vai definir o número de colunas as quais iremos trabalhar.

Para estruturarmos (montarmos) o nosso grid (grade), temos que definir antes qual será o formato que vamos utilizar.

Por exemplo: 4 columns X 2 rows (4 colunas x 2 linhas) - que é o que vamos utilizar nesse exemplo. Formato definido de 4 colunas e 2 linhas precisamos usar a propriedade grid-template-columns para que assim possamos definir a quantidade de colunas e a largura de cada uma delas.

Essa é a estrutura base para definirmos o display como grid e que vamos usar o grid-template-columns:

```
.container {  
    display: grid;  
    grid-template-columns: "aqui definimos o número de colunas"  
}
```

Agora que sabemos como funciona a estrutura base do código que representa o grid-template-columns (o qual digitaremos dentro do arquivo **style.css**)... bora ver como vai ficar a configuração com valores dentro dele.



# Propriedades do CSS Grid

## Grid-Template-Columns:

Para o nosso Grid Container (aqui representado por container-parede), vamos definir alguns valores para melhor entendimento do exemplo:

```
.container-parede {
```

```
  max-width: 1200px;
```

(aqui definimos uma largura máxima de 1200px do nosso container)

```
  display: grid;
```

(aqui habilitamos o uso do display grid)

```
  color: white;
```

(configuramos a cor da fonte como branca)

```
  gap: 10px;
```

(definimos um gap de 10px - lembra do cuidado com o "vão" entre o trem e a plataforma, pois bem aqui estamos definindo esse vão o que será melhor visualizado quando renderizarmos o código no navegador)

```
  background-color: grey;
```

(Configuramos a cor do fundo da nossa parede como cinza, o que vai facilitar o entendimento do gap de 10px configurado acima).

```
  grid-template-columns: 150px 250px 350px 450px;
```

(Aqui definimos 4 colunas e cada coluna recebeu um valor manual)

Coluna 1: 150px

Coluna 2: 250px

Coluna 3: 350px;

Coluna 4: 450px

```
}
```



/igor-rebolla

# Propriedades do CSS Grid

## Grid-Template-Columns:

Dentro do arquivo index.html (criamos 4 div's) com 4 classes diferentes, o que nos permite trabalhar cada uma com exclusividade:

```
.item-revestimento01  
.item-revestimento02  
.item-revestimento03  
.item-revestimento04
```

Só que nesse caso específico eu gostaria que mesmo essas div's tendo classes diferentes fossem definidos os mesmos valores para elas.

Dentro do nosso arquivo style.css (Só colocar essas classes separadas por vírgulas). Bora ver como isso fica:

```
.item-revestimento01, .item-revestimento02, .item-revestimento03,  
.item-revestimento04 {  
    background-color: orange;  
    display: grid;  
    justify-items: center;  
    align-items: center;  
}
```

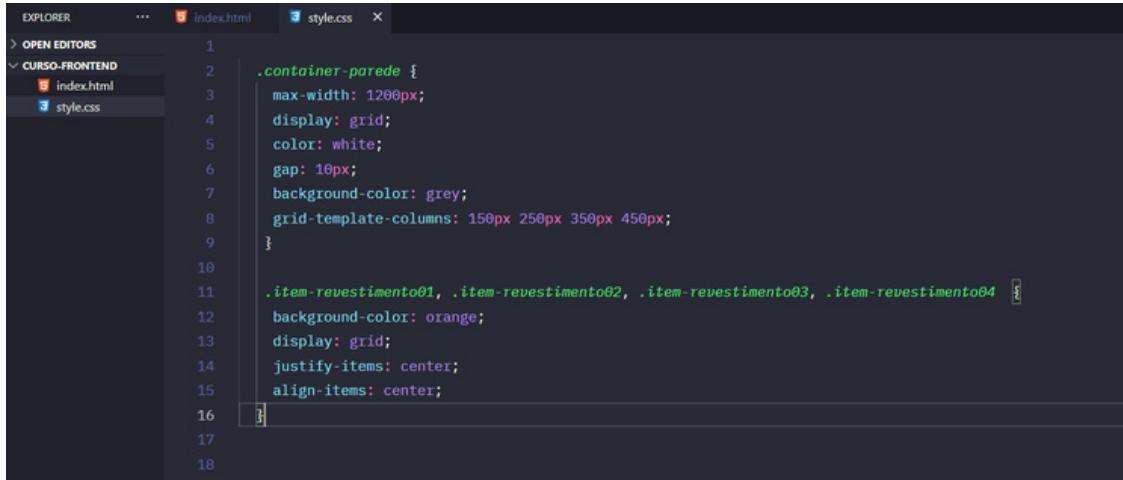
Resumindo: as 4 classes vão receber uma cor de fundo (laranja), habilitamos o display grid aqui também, e os itens (tijolinhos) serão tanto alinhados quanto justificados ao centro.



# Propriedades do CSS Grid

## Grid-Template-Columns:

Vamos ver como ficou esse código dentro do nosso arquivo style.css



```
EXPLORER      index.html  style.css
OPEN EDITORS
CURSO-FRONTEND
index.html
style.css

1 .container-parede {
2   max-width: 1200px;
3   display: grid;
4   color: white;
5   gap: 10px;
6   background-color: grey;
7   grid-template-columns: 150px 250px 350px 450px;
8 }
9
10
11 .item-revestimento01, .item-revestimento02, .item-revestimento03, .item-revestimento04
12   background-color: orange;
13   display: grid;
14   justify-items: center;
15   align-items: center;
16 }
17
18
```

Vamos ver como esse código ficou renderizado no navegador:



- 1 - Representa nossa primeira coluna a qual foi definida com uma largura de 150px
- 2 - Representa nossa segunda coluna a qual foi definida com uma largura de 250px
- 3 - Representa nossa terceira coluna a qual foi definida com uma largura de 350px
- 4 - Representa nossa quarta coluna a qual foi definida com uma largura de 450px

A somatoria das 4 colunas nesse exemplo aqui... totalizam: 1200px

Entre as colunas notem um vão em cinza (esse é o gap de 10px que definimos no .container-parede)



# Propriedades do CSS Grid

## Grid-Template-Columns:

Tá Igor, e tem como esses valores das 4 colunas serem definidos de forma automática e se eu posso definir 3 valores manuais e um de forma automatica?

Tem sim é e exatamente isso que veremos logo abaixo.

Vamos manter a estrutura do nosso arquivo index.html e focar aqui só no style.css.

Para definirmos as nossas colunas todas como automáticas ou seja com larguras iguais. Vamos definir o valor do grid-template-columns, como auto.

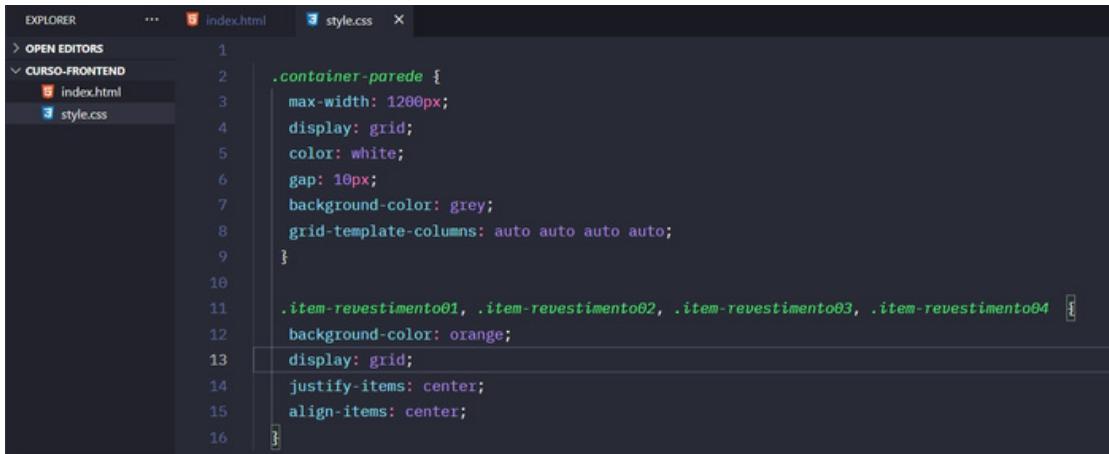
```
.container-parede {  
    max-width: 1200px;  
    display: grid;  
    color: white;  
    gap: 10px;  
    background-color: grey;  
    grid-template-columns: auto auto auto auto;  
}
```

E bora ver na imagem da próxima página como esse código ficou dentro do style.css:



# Propriedades do CSS Grid

## Grid-Template-COLUMNS:



```
EXPLORER ... index.html style.css
OPEN EDITORS
CURSO-FRONTEND
index.html
style.css

1 .container-parede {
2   max-width: 1200px;
3   display: grid;
4   color: white;
5   gap: 10px;
6   background-color: grey;
7   grid-template-columns: auto auto auto auto;
8 }
9
10 .item-revestimento01, .item-revestimento02, .item-revestimento03, .item-revestimento04
11   background-color: orange;
12
13   display: grid;
14   justify-items: center;
15   align-items: center;
16 }
```

E bora ver como esse código ficou renderizado no navegador:



Como definimos os valores das nossas 4 colunas como (auto auto auto auto) cada coluna recebeu um valor igual. Como o valor do nosso container foi definido em 1200px (aqui cada coluna está agora com 300px). E se tivessemos 2 colunas aqui e elas fossem definidas como (auto auto) cada uma teria 600px e assim sucessivamente...

Bom já vimos:

- As 4 colunas com valores definidos manualmente;
- As 4 colunas com valores automáticos;

Agora vamos ver 3 colunas com valores manuais e uma definida como auto



# Propriedades do CSS Grid

## Grid-Template-Columns:

```
.container-parede {  
    max-width: 1200px;  
    display: grid;  
    color: white;  
    gap: 10px;  
    background-color: grey;  
    grid-template-columns: 200px 50px 300px auto;  
}
```

(Aqui definimos 4 colunas e cada coluna recebeu um valor)

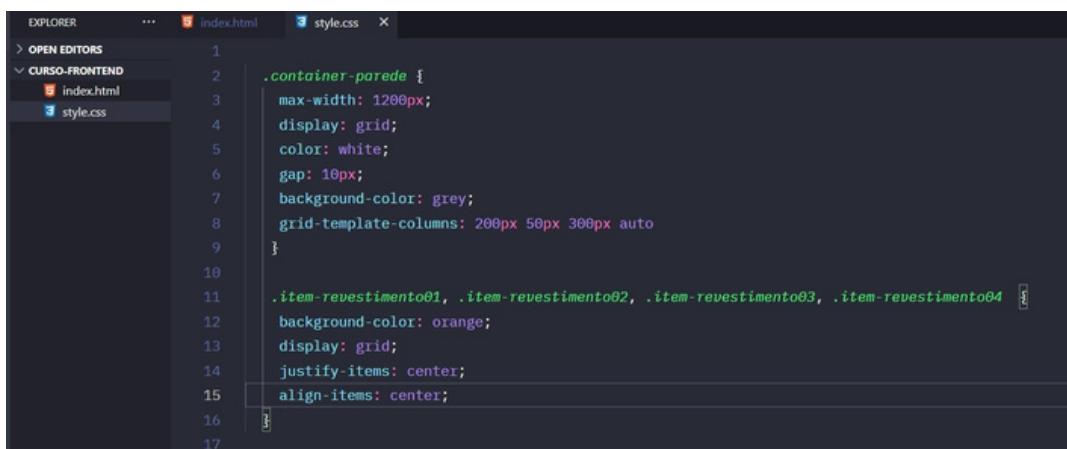
Coluna 1: 200px

Coluna 2: 50px

Coluna 3: 300px

Coluna 4: auto

E bora ver como esse código ficou no style.css:



```
EXPLORER ... index.html style.css  
OPEN EDITORS  
CURSO-FRONTEND  
index.html  
style.css  
1 .container-parede {  
2     max-width: 1200px;  
3     display: grid;  
4     color: white;  
5     gap: 10px;  
6     background-color: grey;  
7     grid-template-columns: 200px 50px 300px auto  
8 }  
9  
10 .item-revestimento01, .item-revestimento02, .item-revestimento03, .item-revestimento04 {  
11     background-color: orange;  
12     display: grid;  
13     justify-items: center;  
14     align-items: center;  
15 }  
16  
17 }
```



/igor-rebolla

# Propriedades do CSS Grid

## Grid-Template-COLUMNS:

E bora ver como esse código será renderizado no navegador:



- 1 - A primeira coluna recebeu o valor de 200px
- 2 - A segunda coluna recebeu o valor de 50px
- 3 - A terceira coluna recebeu o valor de 300px
- 4 - A quarta coluna recebeu o valor de auto

OBS.: Como sugestão eu peço para vocês diminuirem a tela do navegador a qual foi renderizada essa imagem da direita pra a esquerda e vejam que as colunas (1, 2 e 3) não sofrem alterações pois o valor foi definido manualmente já a coluna 4 que recebeu o valor auto vai diminuir conforme sua disposição no container.

Direção da setinha (esquerda para a direita) para ver o comportamento da quarta coluna.



/igor-rebolla

# Propriedades do CSS Grid

## Grid-Template-Rows:

Essa é a propriedade dentro do CSS Grid que vai definir (o número de linhas e sua altura) no Grid as quais iremos trabalhar.

Assim como no grid-template-columns, o grid-template-rows também tem sua configuração base

Essa é a estrutura base para definirmos o display como grid e que vamos usar o grid-template-rows:

```
.container {  
    display: grid;  
    grid-template-rows: "aqui definimos o número de linhas"  
}
```

r

Agora que sabemos como funciona a estrutura base do código que representa o grid-template-rows (o qual digitaremos dentro do arquivo style.css)... bora ver como ficará a configuração com 2 valores (um valor de 125px para a primeira linha(row) e outro de 275px para a segunda linha) a terceira e quarta linha(row) permanecem com seus valores padrões (nesse exemplo).

```
.container-parede {  
    max-width: 1200px;  
    display: grid;  
    color: white;  
    gap: 10px;  
    background-color: grey;  
    grid-template-rows: 125px 275px;  
}
```

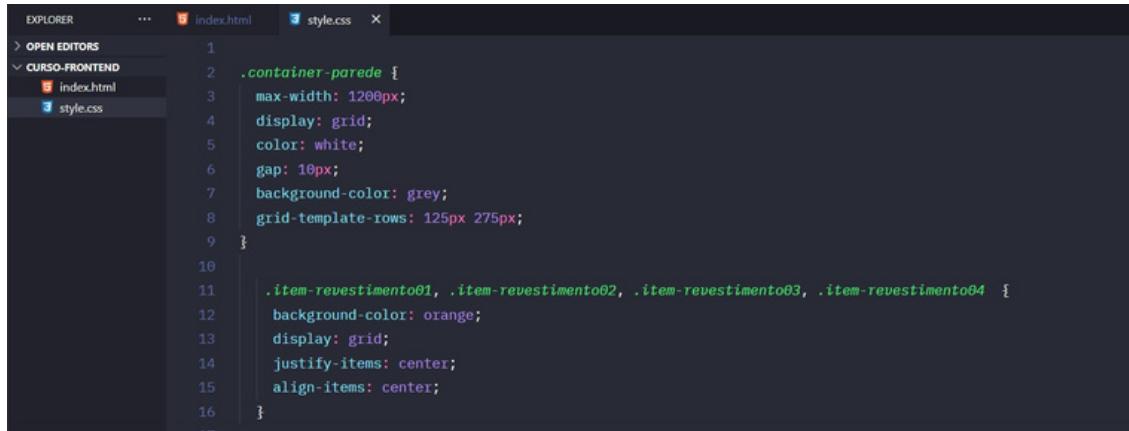


/igor-rebolla

# Propriedades do CSS Grid

## Grid-Template-Rows:

E bora ver como esse código ficou no style.css:



```
EXPLORER    index.html    style.css X
OPEN EDITORS
CURSO-FRONTEND
index.html
style.css

1 .container-parede {
2   max-width: 1200px;
3   display: grid;
4   color: white;
5   gap: 10px;
6   background-color: grey;
7   grid-template-rows: 125px 275px;
8 }
9
10
11 .item-revestimento01, .item-revestimento02, .item-revestimento03, .item-revestimento04 {
12   background-color: orange;
13   display: grid;
14   justify-items: center;
15   align-items: center;
16 }
```

E bora ver como esse código ficou renderizado no navegador:



- A linha correspondente ao Item 01 está com uma altura de 125px
- A linha correspondente ao Item 02 está com uma altura de 275px
- As linhas correspondentes aos itens 03 e 04 estão com sua altura padrão pois não definimos nenhum valor para elas.



# Propriedades do CSS Grid

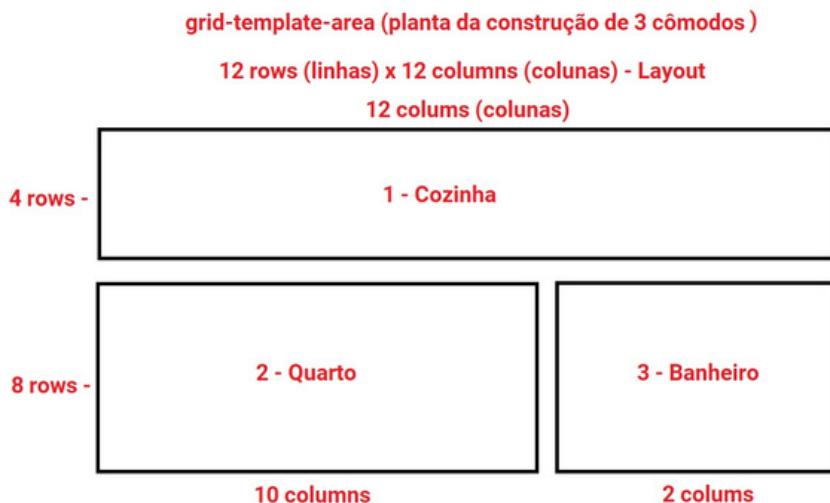
## Grid-Template-Areas:

O grid-template-areas é usado para especificar a quantidade de espaço que uma célula do grid(grade) deve conter em termos de colunas e linhas no container pai.

Para melhor entendimento vamos criar o seguinte layout abaixo tomando como base o exemplo da construção de 3 cômodos em nosso terreno.

- Vamos definir o nosso terreno com uma largura máxima de 800px
- E os cômodos serão: (Cozinha, Quarto, Banheiro).

Com base nessas informações (nossa planta/layout) ficou assim:

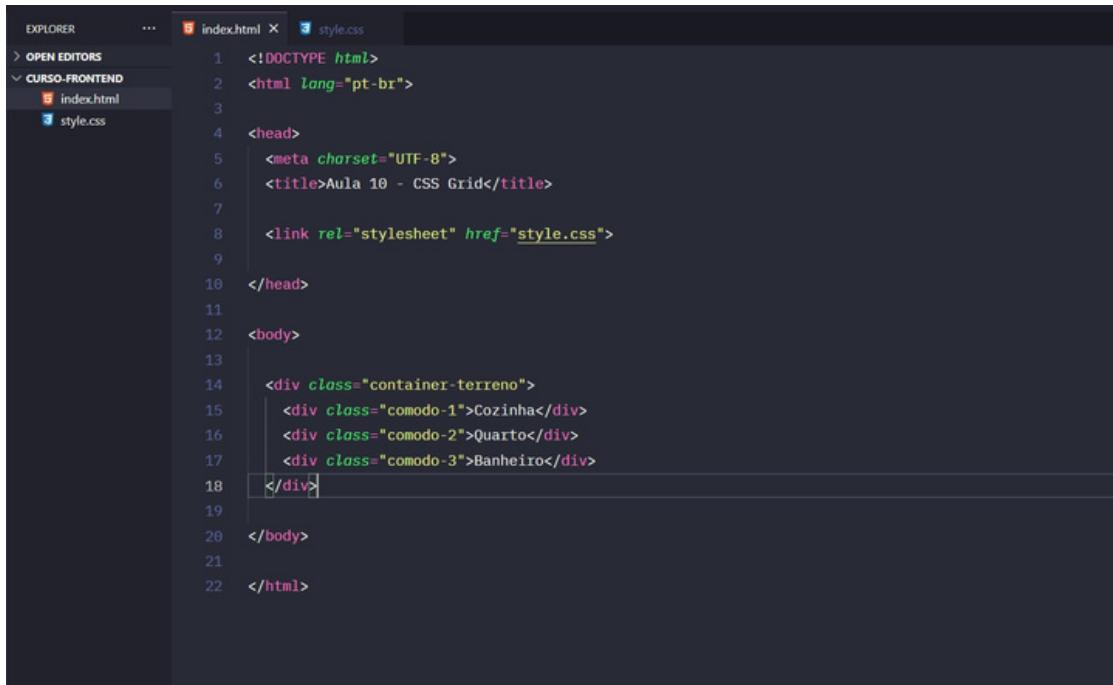


Como temos 3 cômodos, vamos definir essa estrutura no nosso arquivo index.html.



# Propriedades do CSS Grid

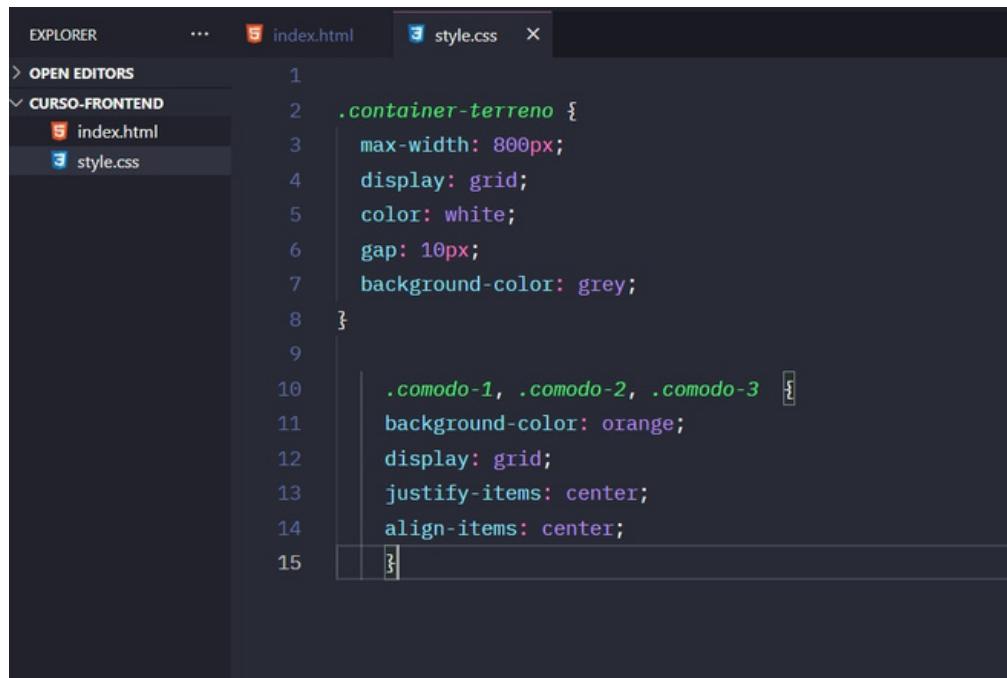
## Grid-Template-Areas:



The screenshot shows a code editor interface with two tabs: 'index.html' and 'style.css'. The 'index.html' tab contains the following HTML code:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Aula 10 - CSS Grid</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container-terreno">
        <div class="comodo-1">Cozinha</div>
        <div class="comodo-2">Quarto</div>
        <div class="comodo-3">Banheiro</div>
    </div>
</body>
</html>
```

Com a estrutura definida no arquivo index.html, vamos definir agora as propriedades do nosso (layout/planta) dentro do arquivo style.css



The screenshot shows a code editor interface with two tabs: 'index.html' and 'style.css'. The 'style.css' tab contains the following CSS code:

```
.container-terreno {
    max-width: 800px;
    display: grid;
    color: white;
    gap: 10px;
    background-color: grey;
}

.comodo-1, .comodo-2, .comodo-3 {
    background-color: orange;
    display: grid;
    justify-items: center;
    align-items: center;
}
```

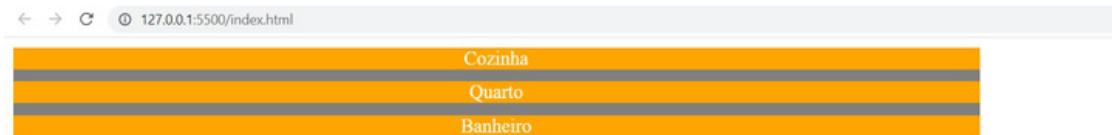


/igor-rebolla

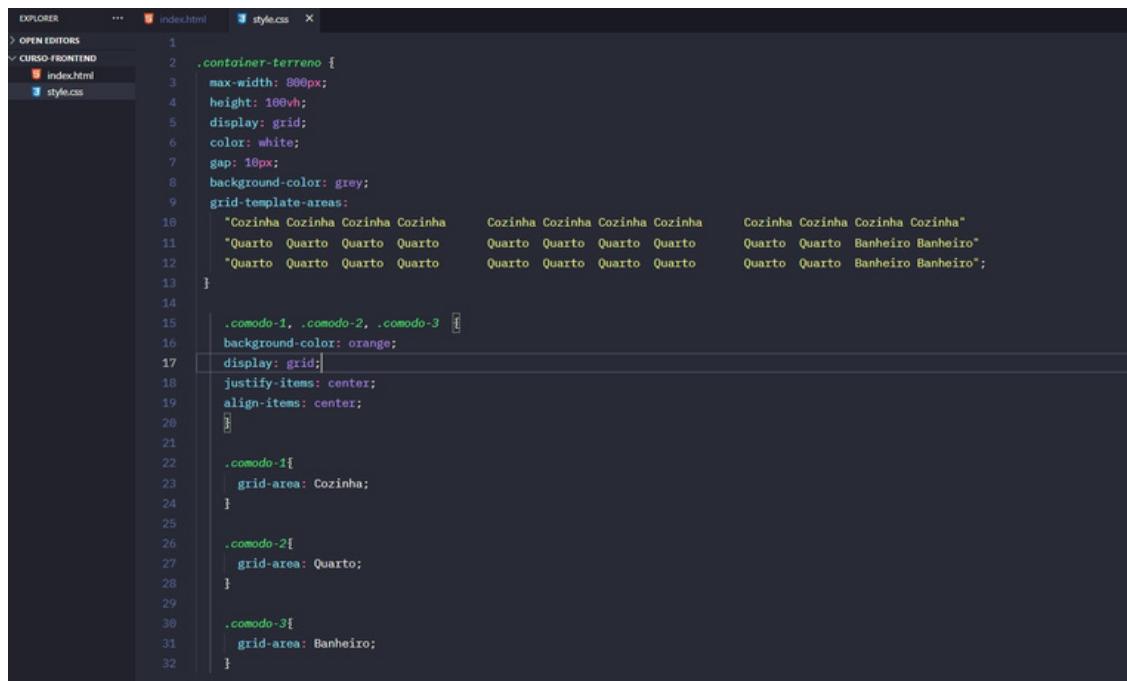
# Propriedades do CSS Grid

## Grid-Template-Areas:

Vamos ver como o código da página anterior ficou renderizado no navegador:



Agora bora utilizar o grid-template-areas. O código dentro do style.css ficará assim:



```
EXPLORER ... index.html style.css
> OPEN EDITORS
  CURSO-FRONTEND
    index.html
    style.css

1 .container-terreno {
2   max-width: 800px;
3   height: 100vh;
4   display: grid;
5   color: white;
6   gap: 10px;
7   background-color: grey;
8   grid-template-areas:
9     "Cozinha Cozinha Cozinha Cozinha"
10    "Quarto Quarto Quarto Quarto"
11    "Quarto Quarto Quarto Quarto"
12    "Quarto Quarto Quarto Quarto"
13 }
14
15 .comodo-1, .comodo-2, .comodo-3 {
16   background-color: orange;
17   display: grid;
18   justify-items: center;
19   align-items: center;
20 }
21
22 .comodo-1{
23   grid-area: Cozinha;
24 }
25
26 .comodo-2{
27   grid-area: Quarto;
28 }
29
30 .comodo-3{
31   grid-area: Banheiro;
32 }
```

Antes da visualização do código acima no navegador, vamos entender o que esse montão de Cozinha, Quarto e Banheiro estão fazendo ai.



# Propriedades do CSS Grid

## Grid-Template-Areas:

No nosso layout/planta da página 17 dessa aula (ficou definido) que nossa Cozinha seria composta por 12 columns x 4 rows. Cada Cozinha que digitamos dentro do grid-template-areas abaixo, representa 1 columns, por isso digitamos 12x a palavra Cozinha e como queremos que essa cozinha tenha 1 altura de 4 linhas (só precisamos digitar 1x essa linha para representar essas 4 linhas).

grid-template-areas:

"Cozinha Cozinha Cozinha"

E para o Quarto e o Banheiro a mesma coisa. Definimos que o quarto seria composto por 10 colunas e 8 linhas e o Banheiro de 2 colunas e 8 linhas. É exatamente isso que o código abaixo está representando 10x a palavra Quarto escrita ( o que representa as 10 colunas) e 2x a palavra Banheiro escrita ( o que representa as 2 colunas). Notem que aqui a linha foi repetida 2x ou seja (Digitamos 20x a palavra Quarto e 4x a palavra Banheiro), isso significa que teremos 8 linhas.

"Quarto Quarto Quarto Quarto Quarto Quarto Quarto Quarto Quarto Quarto Banheiro Banheiro"

"Quarto Quarto Quarto Quarto Quarto Quarto Quarto Quarto Quarto Quarto Banheiro Banheiro";



/igor-rebolla

# Propriedades do CSS Grid

## Grid-Template-Areas:

```
.comodo-1{  
    grid-area: Cozinha;  
}
```

```
.comodo-2{  
    grid-area: Quarto;  
}
```

```
.comodo-3{  
    grid-area: Banheiro;  
}
```

E aqui para cada cômodo especificado, temos que usar o grid-area com o nome do respectivo cômodo para fazer uma espécie de assinatura/verificação no grid-template-areas.

Se um quarto cômodo (por exemplo: lavanderia) fosse adicionado na nossa planta/layout, bastaria especificar esse cômodo dentro do grid-template-areas e assim feito criar um novo grid-area: lavanderia e fazer sua assinatura conforme exemplo abaixo:

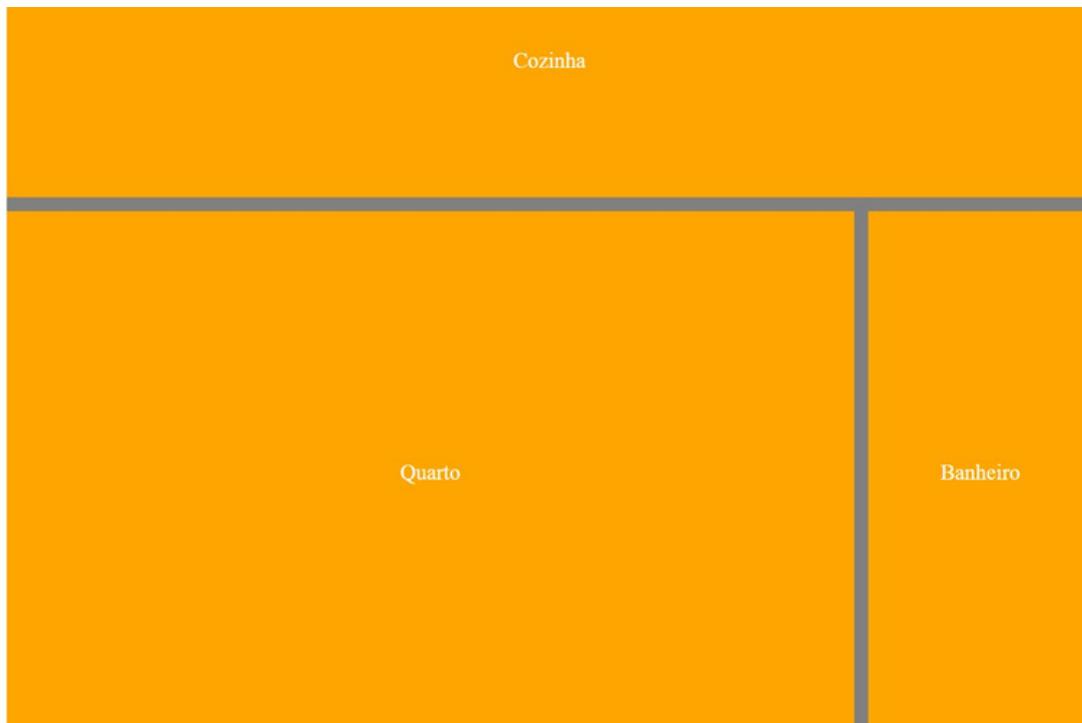
```
.comodo-4{  
    grid-area: Quintal;  
}
```

Agora sim podemos ver o código renderizado no navegador:

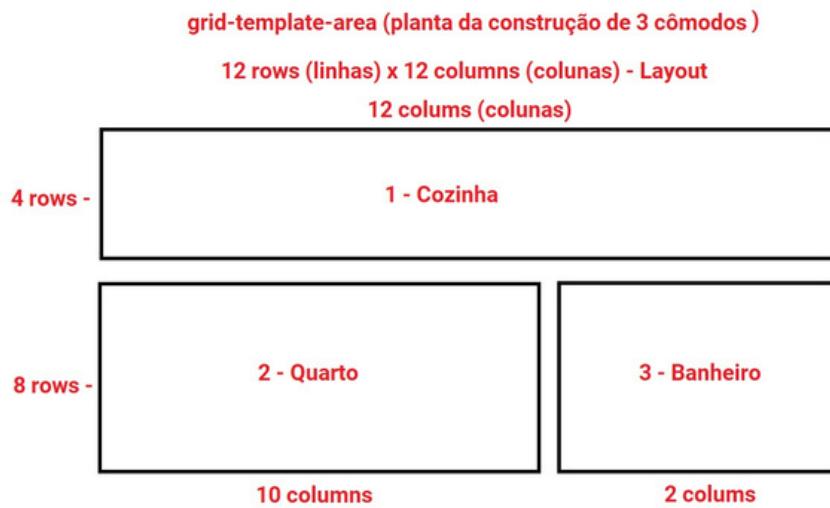


# Propriedades do CSS Grid

## Grid-Template-Areas:



Acima podemos ver o resultado final do nosso layout/planta (3 Cômodos). E abaixo o layout/planta que foi definido na página 17 dessa aula.



/igor-rebolla

# **Sugestões de prática para essa aula:**

1. Abrir o Microsoft Visual Studio Code, depois a pasta Curso-FrontEnd e os arquivos: index.html e style.css;
2. Refaça os exemplos dessa aula;
3. Exemplos feitos (Conforme item 2) bora criar o seu próprio layout/planta – primeiro com 3 cômodos e depois com 4 cômodos.
4. Dentro da imagem da página 19 dessa aula (tem uma propriedade height com um valor definido de 100vh. Pesquisar o que esse VH faz no CSS.

Programação é prática, curiosidade e repetição!!!!



# O que é o Git?

O Git é um sistema de versionamento (gratuito) que armazena e grava mudanças feitas em nossos arquivos de forma local(repositório local) ou seja é como se fosse um depósito na sua obra onde você guarda alguns materiais para que a obra possa ser concluída com aquilo que você tem armazenado localmente(repositório local) sem a necessidade de ir buscar qualquer material no depósito externamente.

Agora que fizemos esse paralelo com a obra, vamos entender como isso funciona com os nossos arquivos e pastas no computador.

Na primeira aula desse curso foi sugerido a criação de uma pasta(local) a qual chamamos de (Curso-FrontEnd) e dentro dessa pasta criamos os arquivos (index.html e o style.css) e nas aulas seguintes esses 2 arquivos passariam por mudanças devido aos conteúdos novos que aprendemos.

Na obra ao guardarmos todo o material localmente, nós corremos o risco de perder uma parte desse material, por exemplo: 5 sacos de cimento que estavam armazenados em um canto no terreno (choveu e ventou bastante) e acabou molhando o cimento, o qual resultou na perda do material.

Agora você fez um site para um cliente, deixou todos os arquivos desse site (que estava quase pronto para ser entregue) armazenados localmente no seu computador e por alguma razão o computador "deu pane"... resultando na perda de todos os arquivos e não tinha sido feito backup.

Aliás tinha sido feito backup sim, localmente na máquina e salvo em pastas da seguinte maneira:



# O que é o Git?

1. Versão 01;
2. Versão 02;
3. Versão Agora vai;
4. Versão uhuuuu Falta pouco;
5. Versão Final para o cliente.

E por estarem armazenadas em repositório local (todas essas versões foram perdidas)

Por isso o recomendado é guardar versões anteriores dos seus projetos para caso você precise voltar e fazer uma alteração específica na Versão 02 por exemplo. Só que notem, mesmo em um exemplo com "apenas" 5 pastas diferentes acima, fica um pouco complicado organizar essas versões e para isso podemos instalar no nosso computador um carinha chamado "Gerenciador de Versões", o Gerenciador de Versões possibilita que para cada versão que fizermos (e essas versões sofrem alterações) eu falo: - 'Amigão Gerenciador de Versões, isso aqui é uma nova versão' e ele gentilmente responde: "OK, versão nova guardada/armazenada".

Estou na aula 02 desse curso, criei os exemplos dessa aula respectiva, fui para a aula 03 estava lá todo "empolgadão" criando os exemplos dessa aula 03 e lembrei "Não curti algo que foi feito na aula 02" agora que o Gerenciador de Versões está instalado, eu posso falar para ele "Gerenciador de Versões, eu preciso alterar o arquivo da Aula 02" como ele guarda/armazena essas versões você pode acessar essa versão e fazer as alterações necessárias.

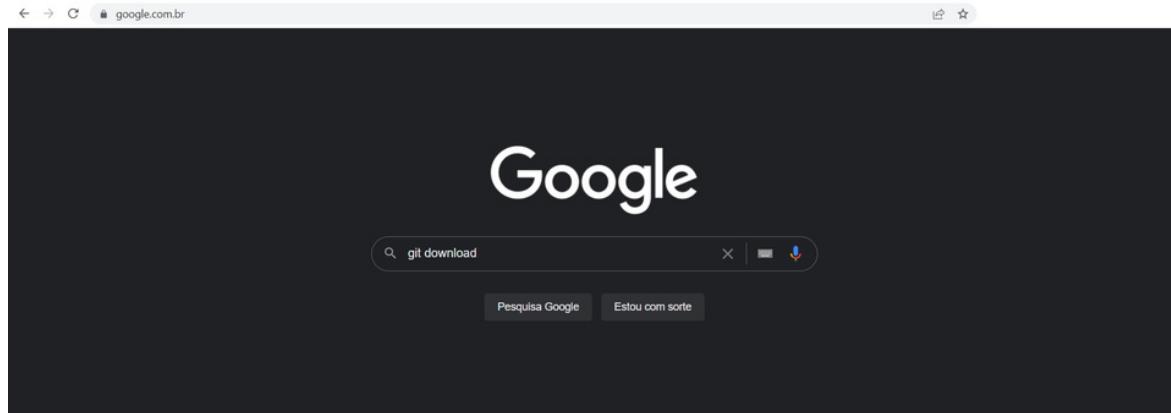
Só reforçando o nome desse Software que faz esse Gerenciamento de Versões é o GIT.

E bora ver como instalamos o GIT.



# Instalação do Git

Passo 1 – Na caixa de pesquisa do Google, vamos digitar git download



Passo 2 – Clicar no primeiro link mostrado com base na pesquisa do Passo 1

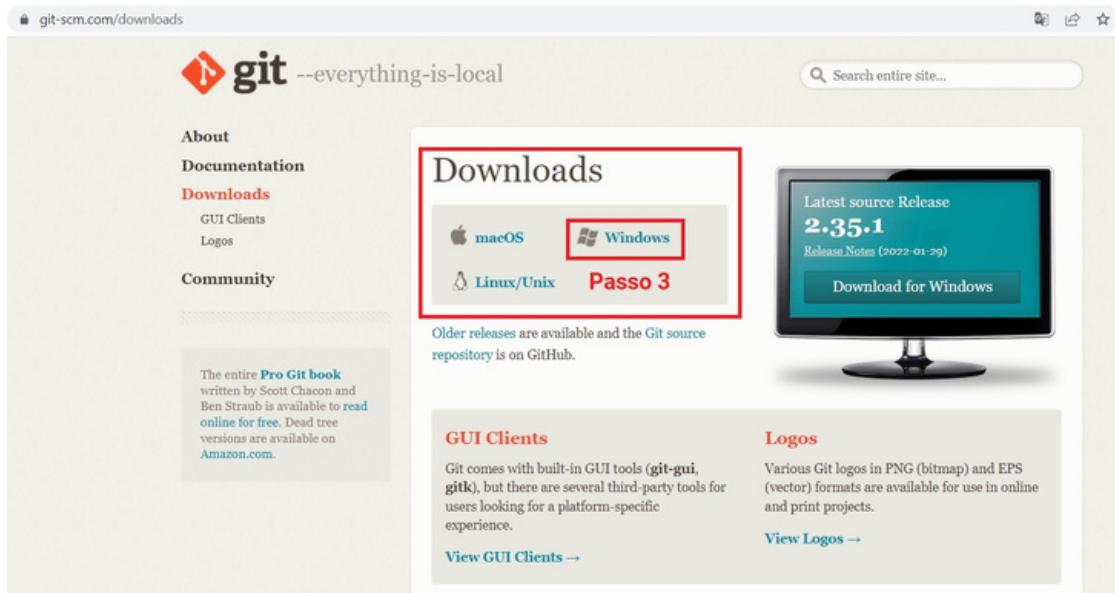
A screenshot of a Google search results page. The search query 'git download' is entered in the search bar. The results show approximately 348,000,000 results. The top result is a link to 'https://git-scm.com' with the title 'Downloads - Git SCM'. This link is highlighted with a red box and labeled 'Passo 2'. Below it, there are other links for 'Downloading Package - Git SCM' and 'Instalando o Git'. The page includes standard Google navigation bars like 'Todas', 'Videos', 'Imagens', 'Notícias', 'Livros', and 'Mais'.



/igor-rebolla

# Instalação do Git

Passo 3 – Como eu estou usando o Windows (vou clicar dentro da Caixa: Downloads – Windows)



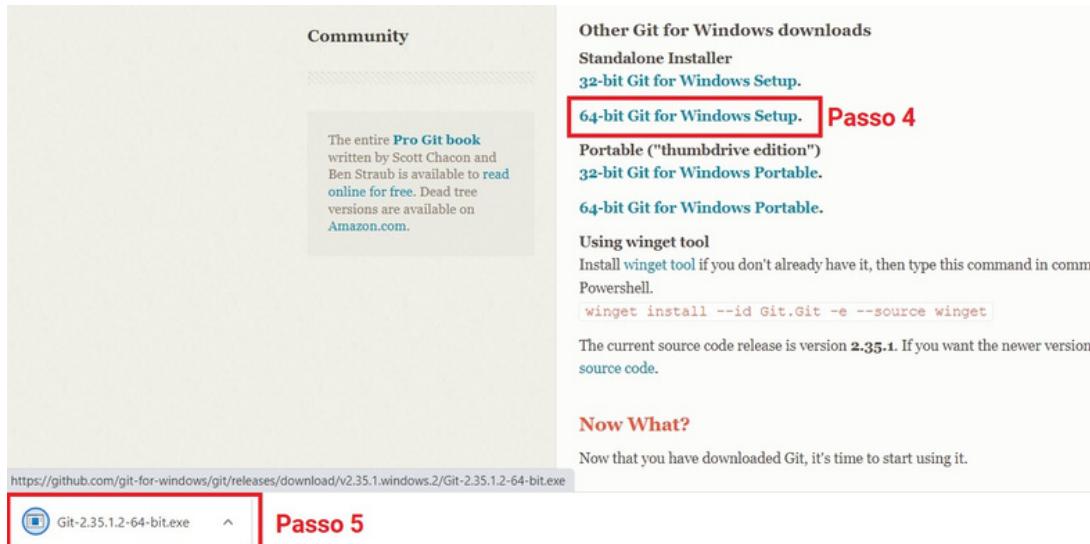
Passo 4 – Aqui eu posso escolher entre as versões do Windows (32-bit e 64-bit) – vou instalar a versão 64-bit



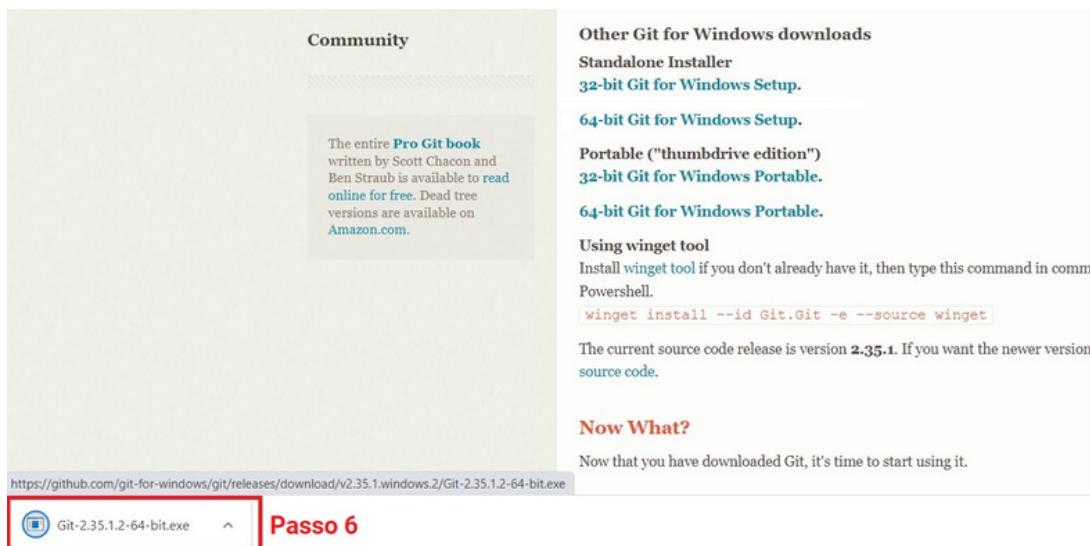
/igor-rebolla

# Instalação do Git

Passo 5 – Ao clicar no passo 4, o download da Versão escolhida do Git vai começar automaticamente.



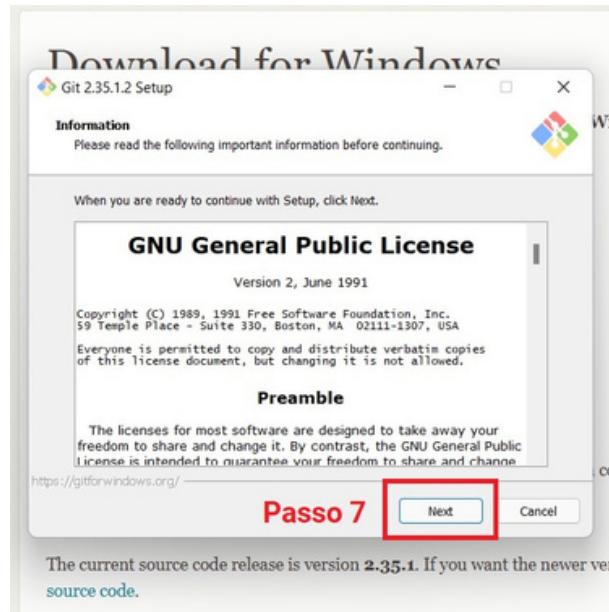
Passo 6 – Download concluído (Clicar 2x sobre o mesmo) e bora seguir os passos da instalação.



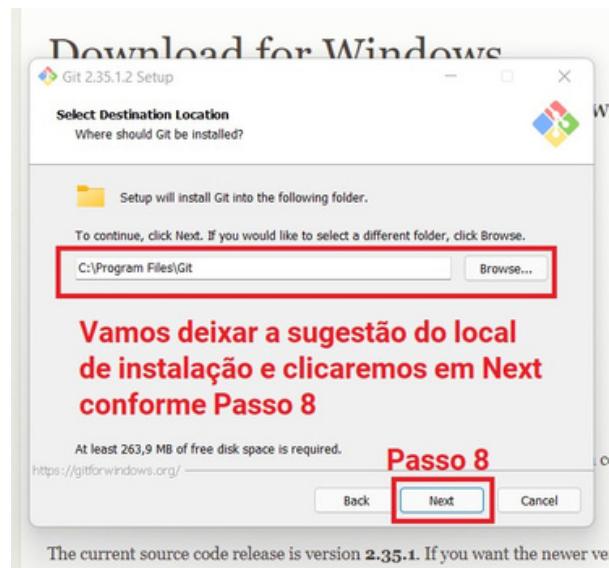
/igor-rebolla

# Instalação do Git

Passo 7 - A primeira caixa que será mostrada é sobre a Licença (Vamos clicar em Next)



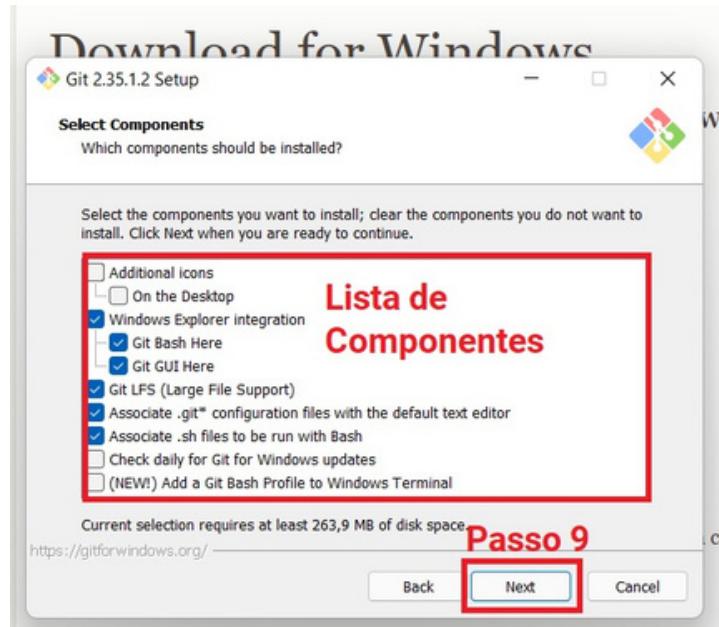
Passo 8 - Aqui vamos escolher o local onde o GIT será instalado no Windows (Vamos deixar a sugestão e clicar em next)



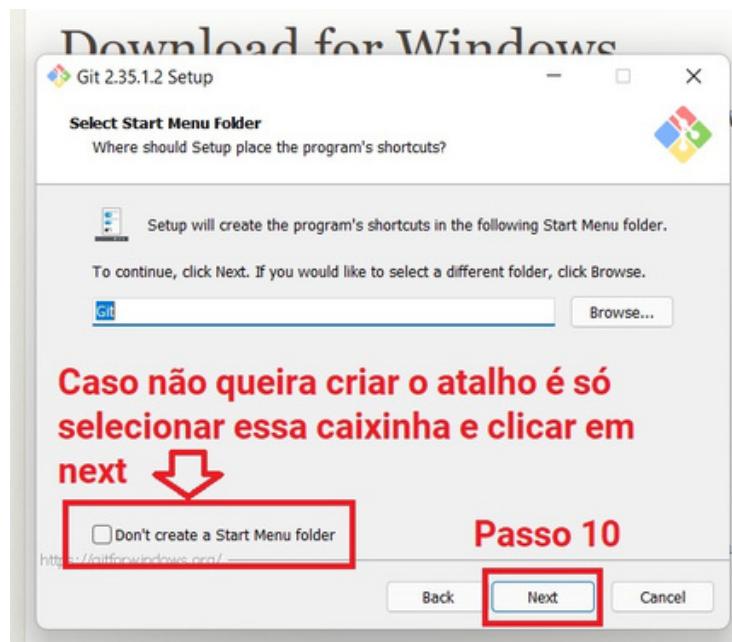
/igor-rebolla

# Instalação do Git

Passo 9 – Essa parte da instalação (permite acrescentar ou tirar alguns componentes), vamos deixar a sugestão e clicar clicar em Next.



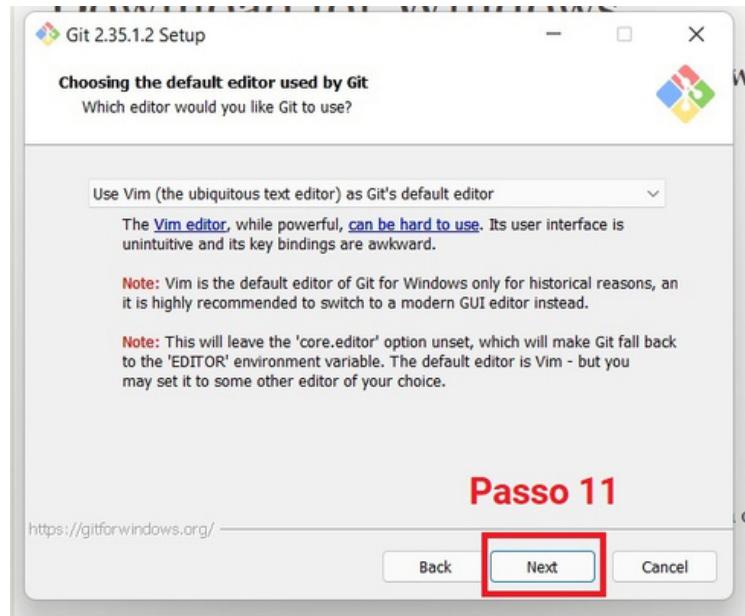
Passo 10 – Permite adicionar ou não um atalho no Menu Iniciar (Vamos adicionar esse atalho)



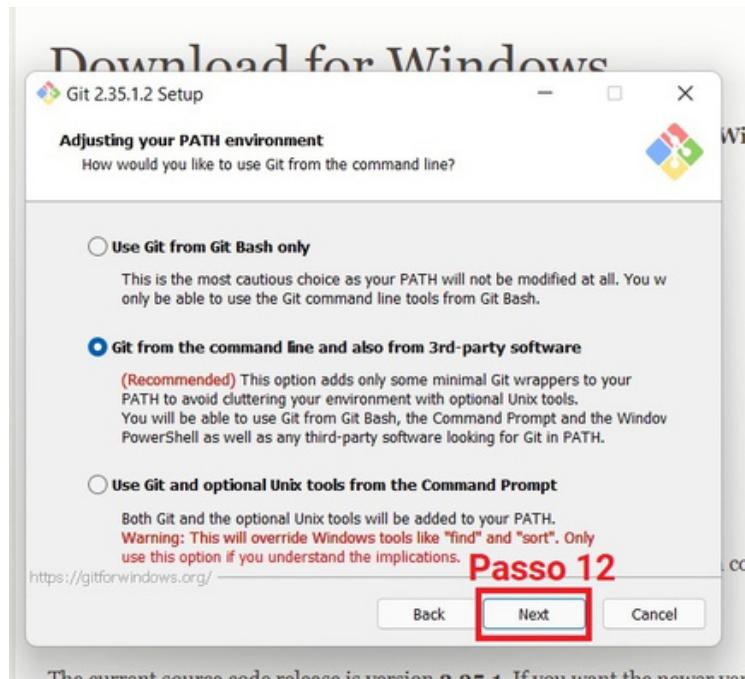
/igor-rebolla

# Instalação do Git

Passo 11 – Está informando qual editor que o Git vai usar (Bora clicar em Next)



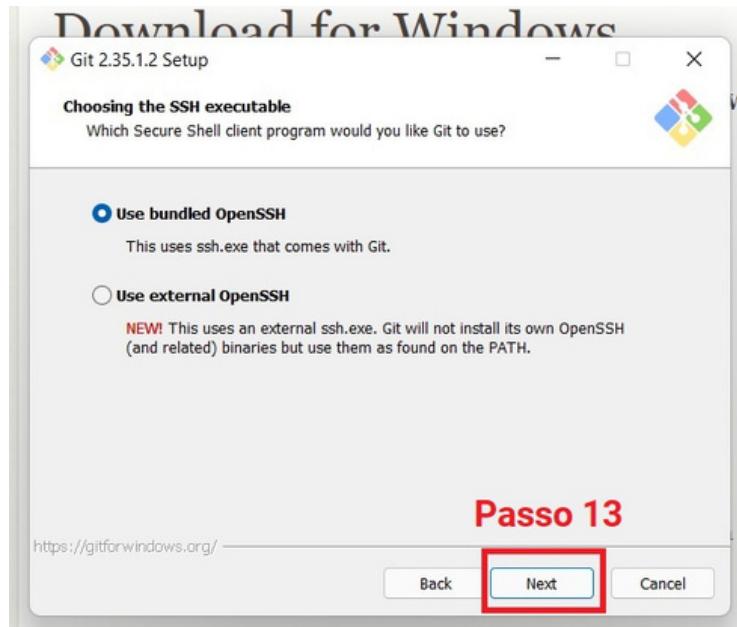
Passo 12 – Vamos seguir a recomendação da instalação e deixaremos selecionada a opção do meio e clicaremos em Next



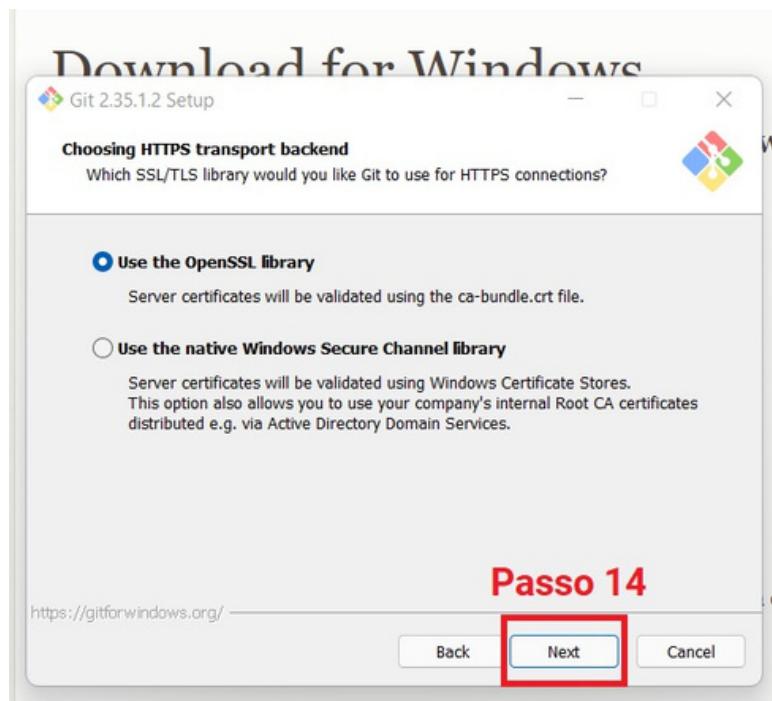
/igor-rebolla

# Instalação do Git

Passo 13 - Vamos seguir a recomendação da instalação e deixaremos selecionada a primeira opção e clicaremos em Next



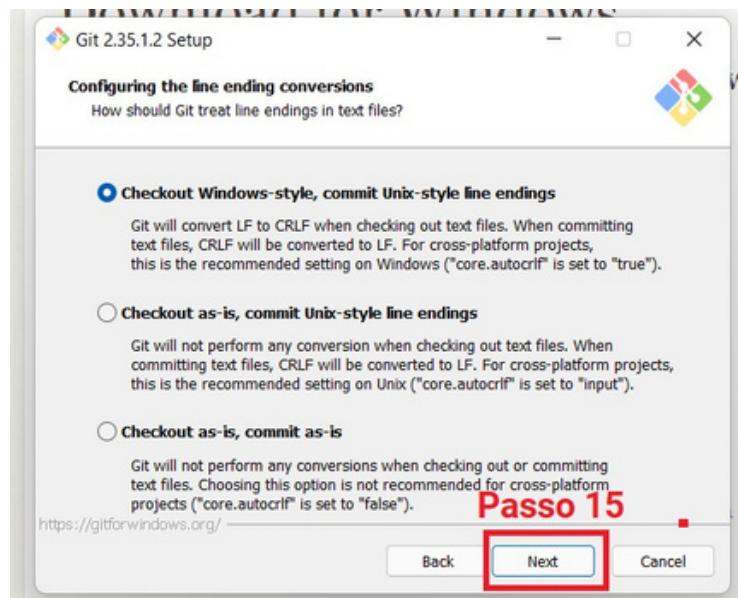
Passo 14 - Vamos seguir a recomendação da instalação e deixaremos selecionada a primeira opção e clicaremos em Next



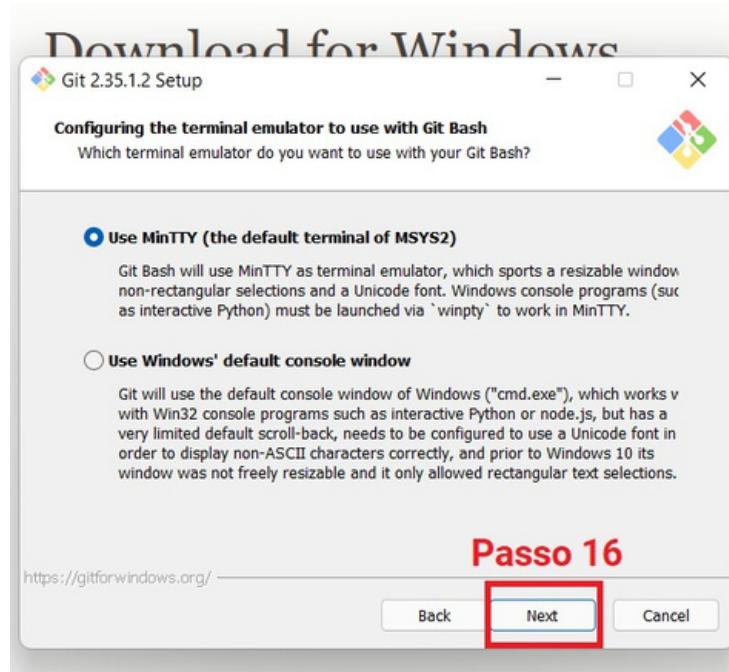
/igor-rebolla

# Instalação do Git

Passo 15 – Vamos seguir a recomendação da instalação e deixaremos selecionada a primeira opção e clicaremos em Next



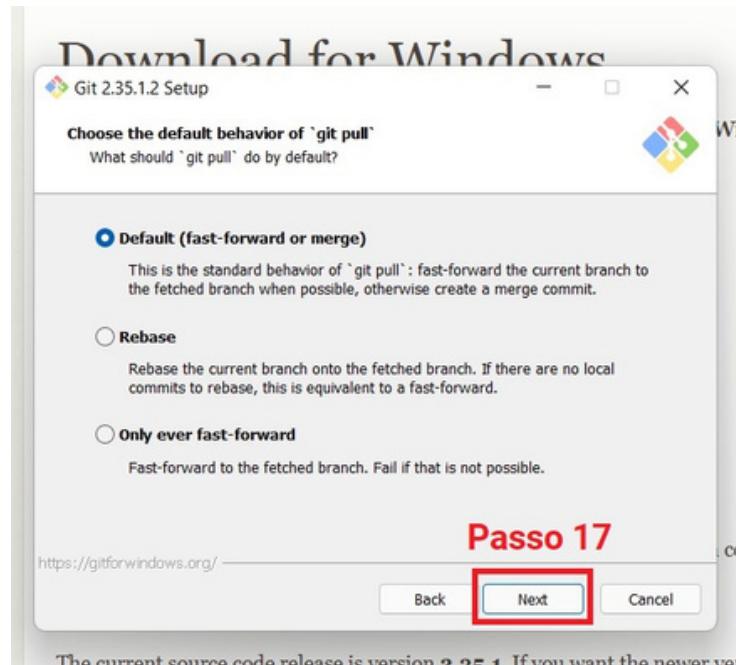
Passo 16 – Vamos seguir a recomendação da instalação e deixaremos selecionada a primeira opção e clicaremos em Next



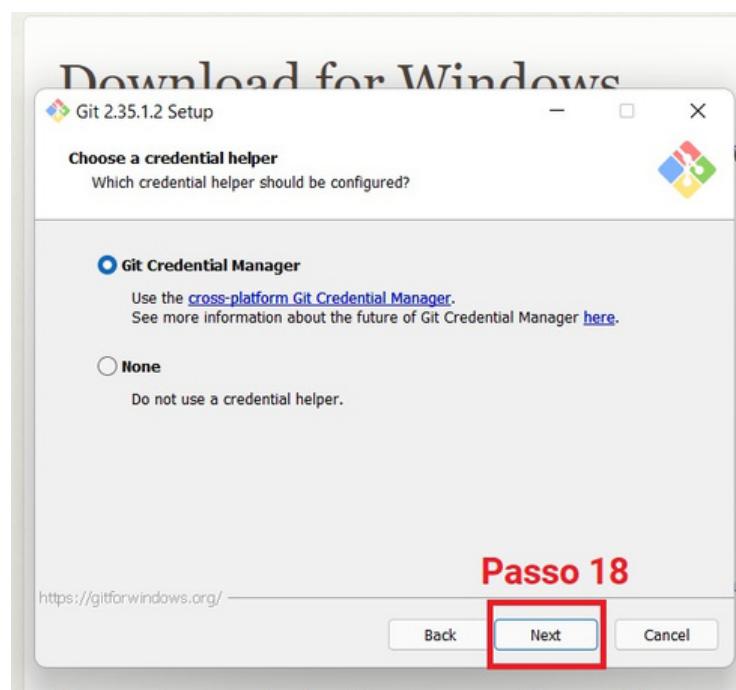
/igor-rebolla

# Instalação do Git

Passo 17 – Vamos seguir a recomendação da instalação e deixaremos selecionada a primeira opção e clicaremos em next



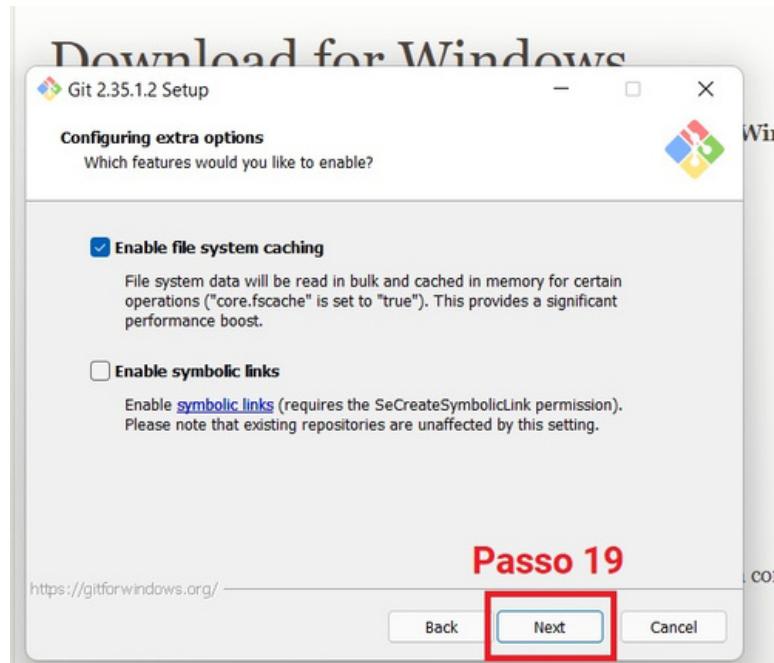
Passo 18 – Vamos seguir a recomendação da instalação e deixaremos selecionada a primeira opção e clicaremos em Next



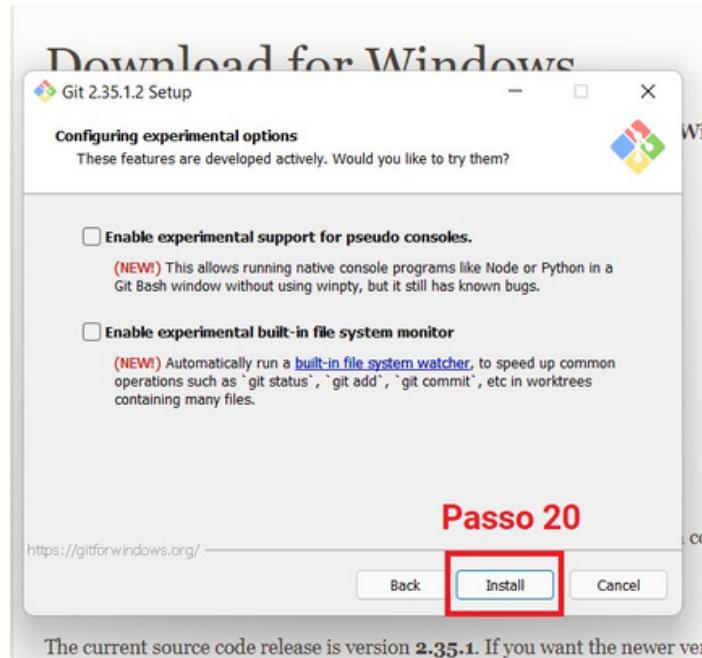
/igor-rebolla

# Instalação do Git

Passo 19 – Vamos seguir a recomendação da instalação e deixaremos selecionada a primeira opção e clicaremos em Next



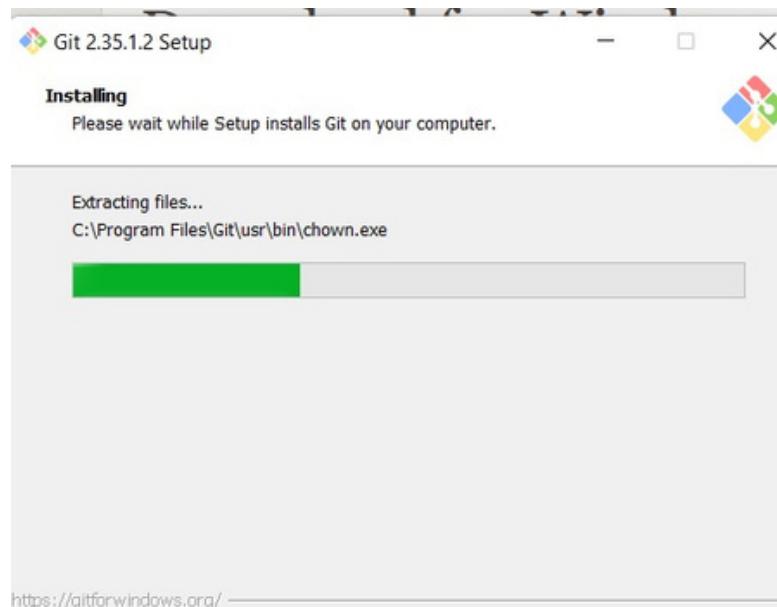
Passo 20 – Aqui não vou selecionar nenhuma opção e vou clicar em Install.



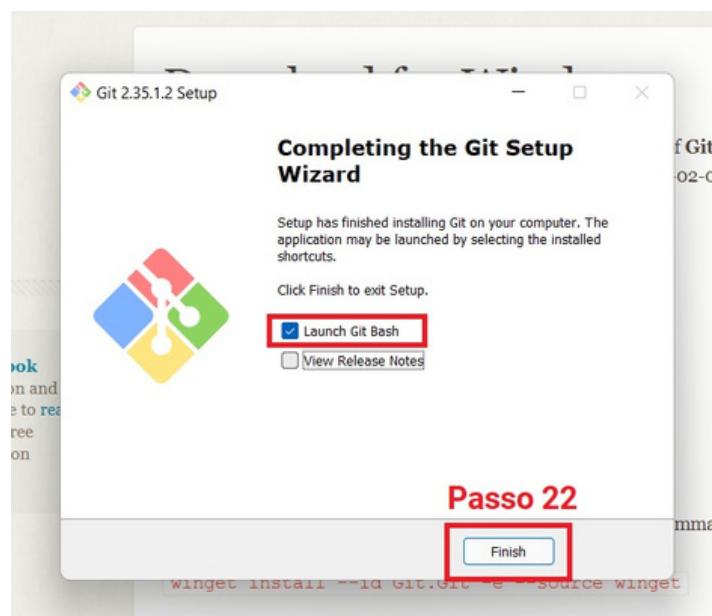
/igor-rebolla

# Instalação do Git

Passo 21 – Vamos aguardar o processo de instalação do GIT



Passo 22 – Instalação Concluida vamos selecionar a opção "Launch Git Bash" e clicaremos em Finish. O Git Bash é um terminal (o qual iremos utilizar aqui nessa aula) do Git. Ele é semelhante ao bom e velho prompt de comando do Windows.



/igor-rebolla

# Instalação do Git

Passo 23 - Quando selecionamos a opção "Launch Git Bash" no passo 22  
(Irá abrir uma tela como essa abaixo)



A screenshot of a Windows desktop environment showing a terminal window titled "Download for Windows". The window is a standard black terminal with white text. The title bar also displays the path "MINGW64:/c/Users/igor\_". The command line shows the user's name "igor\_@DESKTOP-KL8KD7Q MINGW64 ~ (master)" followed by a prompt "\$". Below the prompt, the letters "k", "a", "n", "o", "r", "e", and "l" are visible, likely being typed or pasted. At the bottom of the terminal window, there is a red status bar with the text "winget install --id Git.Git -e --source winget". The background of the desktop is a light blue gradient.



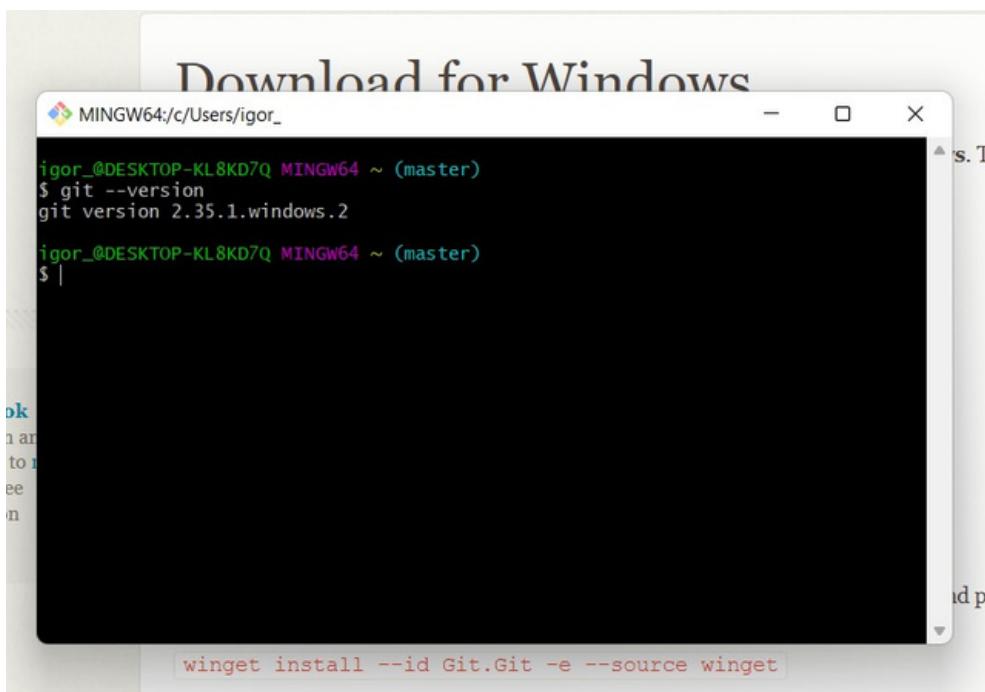
/igor-rebolla

# Instalação do Git

Passo 24 – Depois do primeiro contato com o Git Bash. O primeiro comando que iremos executar até para começarmos a nos acostumar com esses comandos é o:

```
git --version
```

Esse comando mostra qual é a versão do GIT que está instalada. Aqui a versão instalada é a: 2.35.1.windows.2 (Isso significa que nossa instalação está funcionando corretamente).



```
MINGW64:/c/Users/igor_ igor_@DESKTOP-KL8KD7Q MINGW64 ~ (master)
$ git --version
git version 2.35.1.windows.2
igor_@DESKTOP-KL8KD7Q MINGW64 ~ (master)
$ |
```

The screenshot shows a terminal window titled "Download for Windows". The terminal is running on a Windows system (MINGW64). The user has run the command "git --version" and received the output "git version 2.35.1.windows.2". The terminal window has a dark background and light-colored text. The title bar also displays the command "winget install --id Git.Git -e --source winget".



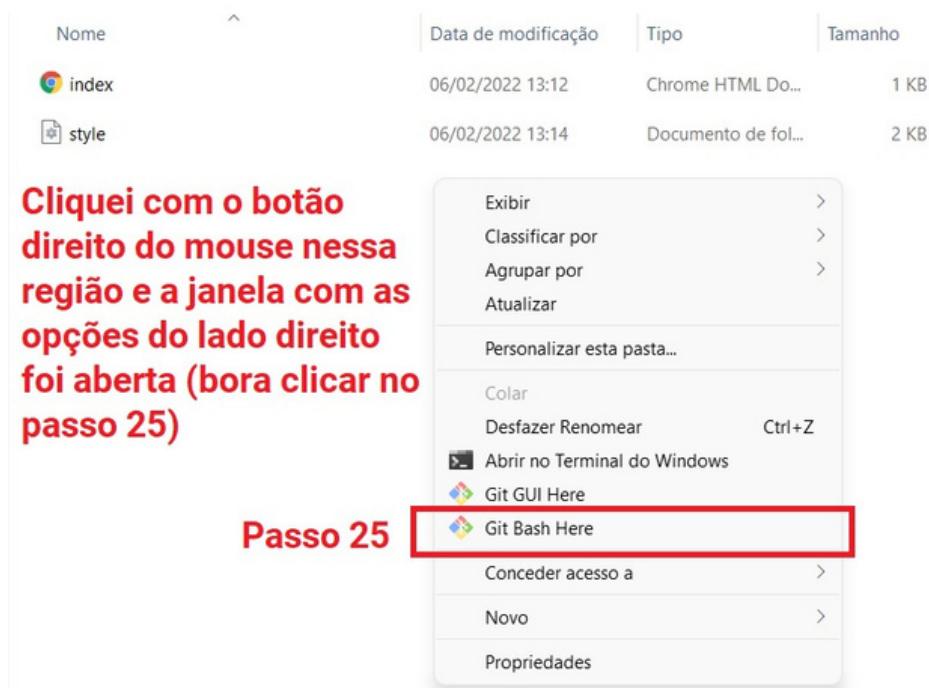
/igor-rebolla

# Instalação do Git

Passo 25 – Vamos procurar o local onde a pasta Curso-FronEnd foi criada.

A minha foi criada no D:\Curso-FrontEnd (vou clicar 2x nessa pasta e aqui é esperado que 2 arquivos tenham sido criados: 1. index.html e 2. style.css).

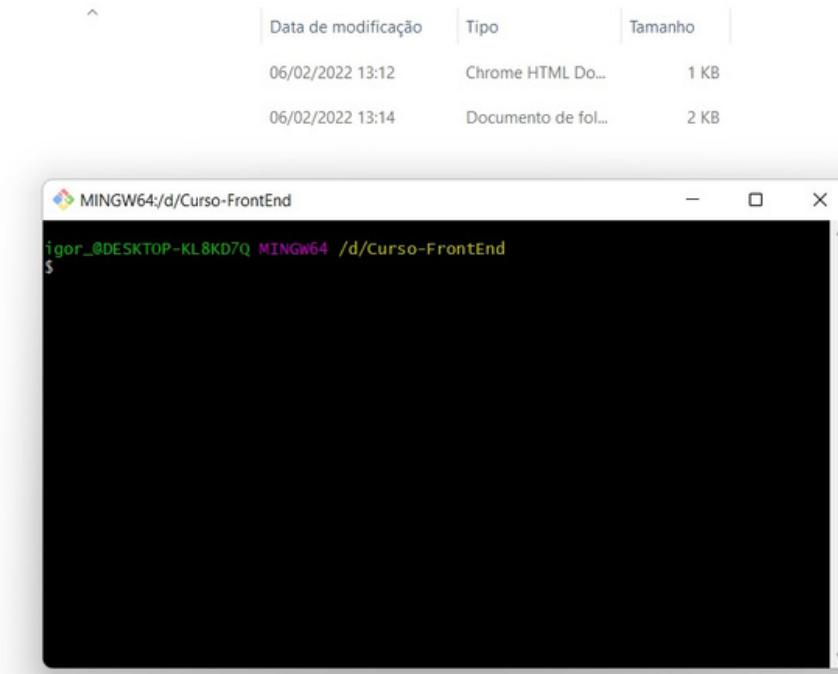
Clicaremos com o botão direito do mouse e podemos ver o Git Bash Here (bora clicar nele)



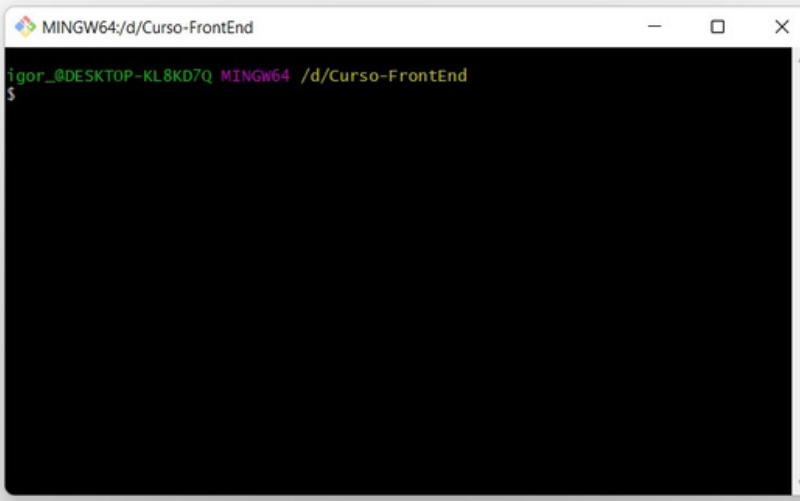
/igor-rebolla

# Instalação do Git

Passo 26 - Com o Git Bash aberto dentro da pasta Curso-FrontEnd essa tela irá surgir e notem que agora está mostrando:  
/d/Curso-FrontEnd pra gente (está mostrando exatamente onde estamos) que é dentro da pasta Curso-FrontEnd.



Nome	Data de modificação	Tipo	Tamanho
index	06/02/2022 13:12	Chrome HTML Do...	1 KB
style	06/02/2022 13:14	Documento de fol...	2 KB

```
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd$
```



# Instalação do Git

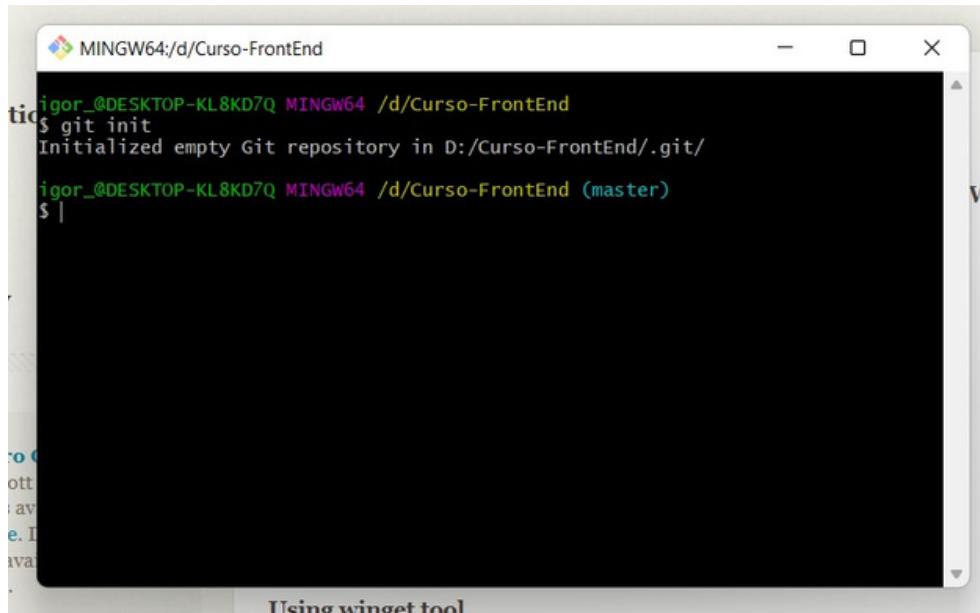
Passo 27 - Agora vamos ver outro comando que é o git init (Esse comando mostra pra gente que iniciamos um repositório git local vazio (empty)).

git init

Ah já ia quase me esquecendo notem que depois que o comando git init foi dado, apareceu um tal de (master).

E qual é a função desse carinha?

É nos mostrar que estamos dentro da (branch master) ou seja da raiz master que quando criamos o repositório ela é criada de forma automática.



```
MINGW64:/d/Curso-FrontEnd
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd
$ git init
Initialized empty Git repository in D:/Curso-FrontEnd/.git/
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (master)
$ |
```



/igor-rebolla

# Instalação do Git

Passo 28 - O git init (inicializou pra gente um repositório vazio) só que na nossa pasta Curso-FrontEnd (temos 2 arquivos dentro dela) o index.html e o style.css. Vamos ver agora como adicionamos esses arquivos dentro do nosso repositório. Esse comando é o:

git add

O git add ele vai mandar os nossos 2 arquivos (index.html e o style.css) para uma especie de "bastidor"... "Bastidor Igor, o que é isso?"

Os arquivos vão ficar aguardando sua vez de serem chamados para entrar em cena (assim como em um espetáculo).

O git add vai convocar esses arquivos que estavam lá aguardando e colocará nessa área de bastidor (o qual você vai ficar aguardando até ser "empurrado" vai fazer sentido já já essa palavra "empurrado") para a parte principal.

Aqui como eu vou adicionar 2 arquivos, vou usar o:  
git add .

O .(ponto) aqui permite que eu adicione todos os arquivos ao mesmo tempo. Caso eu queira adicionar só um arquivo é só usar: git add index.html e só o index.html será adicionado o style.css não.

```
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd
$ git init
Initialized empty Git repository in D:/Curso-FrontEnd/.git/
Pr
id
re
te
m
$ |
Now What?
```



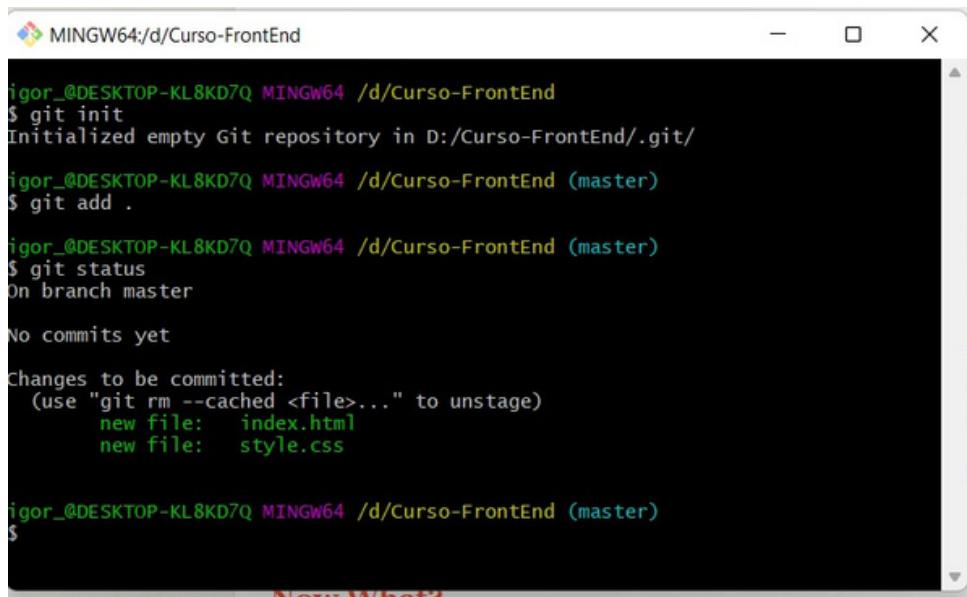
# /igor-rebolla

# Instalação do Git

Passo 29 – Notem que parece que nada aconteceu, foi dado o git add . e ele simplesmente foi para a linha da nossa pasta. Mas aconteceu sim e aqui vamos usar um novo comando que é o:

```
git status
```

Esse comando mostra os arquivos que foram adicionados naquele "bastidor" pelo git add (e os quais ainda não foram comitados).



```
MINGW64:/d/Curso-FrontEnd
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd
$ git init
Initialized empty Git repository in D:/Curso-FrontEnd/.git/
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (master)
$ git add .

igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (master)
$ git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file:   index.html
  new file:   style.css

igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (master)
$
```



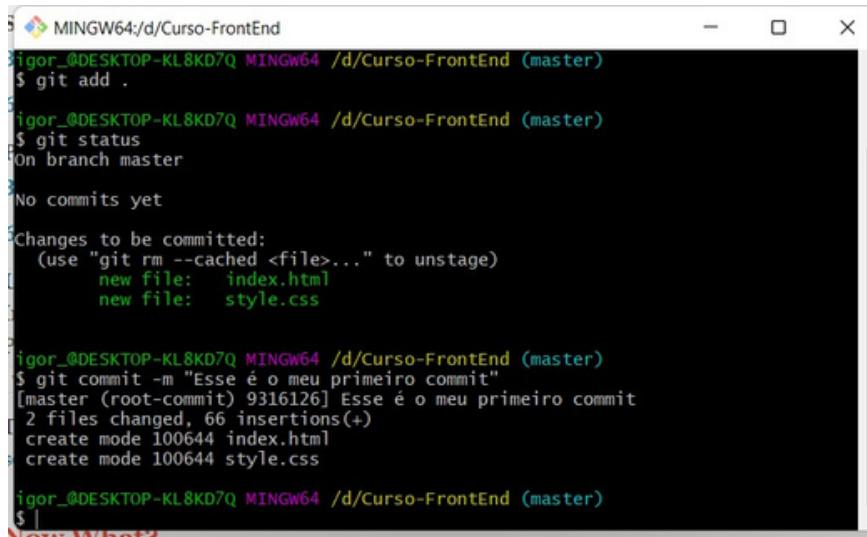
# Instalação do Git

passo 30 - O comando que usaremos para fazer o commit é o:

```
git commit -m "Esse é o meu primeiro commit"
```

Os commits são as versões dos nossos arquivos.

Dado esse comando e conforme imagem abaixo que nos mostra que conseguimos fazer o primeiro commit (primeira versão) em nosso repositório.



The screenshot shows a terminal window titled 'MINGW64/d/Curso-FrontEnd'. The user has run several commands:

```
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (master)
$ git add .
.
.
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (master)
$ git status
On branch master
.
.
No commits yet
.
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   index.html
    new file:   style.css
.
.
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (master)
$ git commit -m "Esse é o meu primeiro commit"
[master (root-commit) 9316126] Esse é o meu primeiro commit
  2 files changed, 66 insertions(+)
  create mode 100644 index.html
  create mode 100644 style.css
.
.
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (master)
$ | Now What?
```

Obs.: O git alterou a nomenclatura da branch principal que vimos em alguns passos anteriores que é chamada de master para main.

"Então porque você não mostrou o main direto aqui?"

Algumas empresas ainda estão utilizando o master... e foi por isso que eu mostrei o main antes. Como esse curso é a base... o ideal é mostrar tudo gradativo (com base no que acontece no mercado de trabalho).

O comando para alterar a branch de master para main é o:

```
git branch -M "main"
```



# Instalação do Git

Vamos ver como fica essa alteração dentro do Git Bash:

```
MINGW64:/d/Curso-FrontEnd
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (master)
$ git status
On branch master
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   index.html
    new file:   style.css

igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (master)
$ git commit -m "Esse é o meu primeiro commit"
[master (root-commit) 9316126] Esse é o meu primeiro commit
 2 files changed, 66 insertions(+)
 create mode 100644 index.html
 create mode 100644 style.css

igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (master)
$ git branch -M "main"
$ Now What?
```

Agora mesmo com o Git instalado na máquina (estou com as minhas versões todas "gerenciadas") e por mais que os arquivos estejam todos versionados (eles continuam armazenados em um repositório local). E se eu quiser:

- mostrar esses arquivos que eu estou estudando para mais pessoas;
- entregar todos os arquivos ou ir mostrando as versões durante o seu desenvolvimento para o meu cliente;
- deixar salvo em algum lugar(repositório remoto) para caso meu computador tenha dado problema (eu consiga buscar essas versões)



# O que é o GitHub?

Para isso tem o Github que é um repositório remoto que permite armazenar seus arquivos e assim (salvar, compartilhar....) tudo de forma segura.

O GitHub permite também que conforme vamos armazenando nossos projetos, seja de (estudo, clientes...) vamos gerando um portfólio para quando formos nos candidatar para alguma vaga de emprego (o recrutador vai pedir o seu link do github) para verificar o que você desenvolveu até aquele momento.

Outra coisa importante sobre o GitHub é que ele permite que trabalhemos em equipe (Vou deixar a sementinha plantada na cabeça de vocês) sobre esse tema.

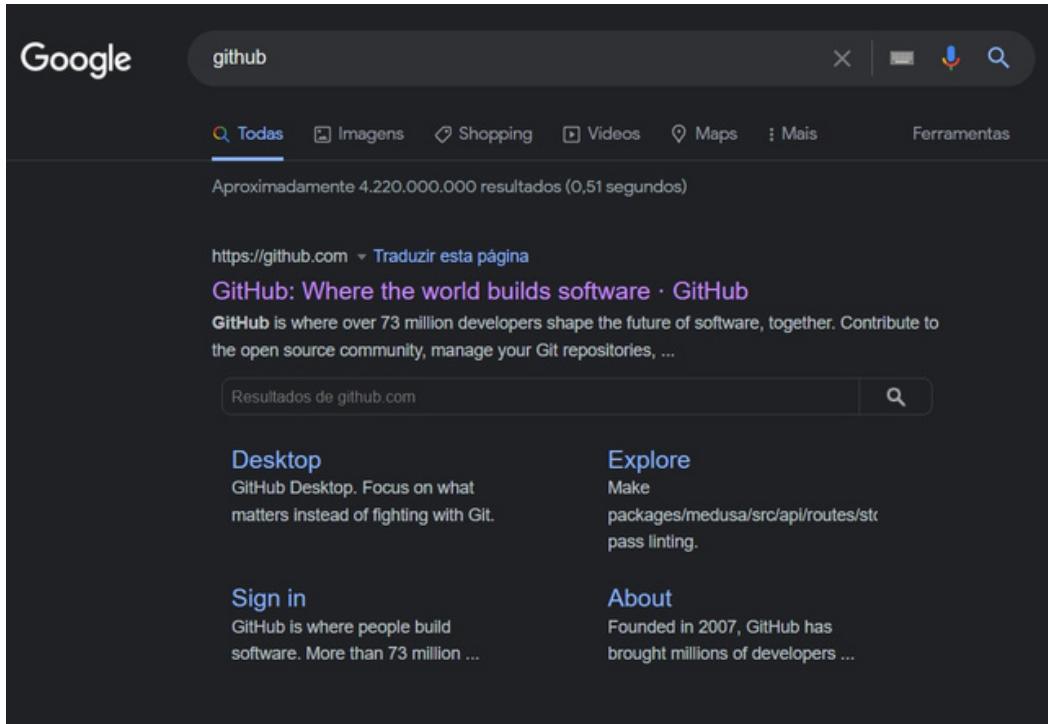
**Resumindo: O GIT vai armazenar versões dos arquivos dentro do computador (repositório local) e o GitHub vai fazer uma cópia desses arquivos/pastas na nuvem (repositório remoto).**

Bora ver como é feito a instalação do GitHub...



# Instalação do GitHub

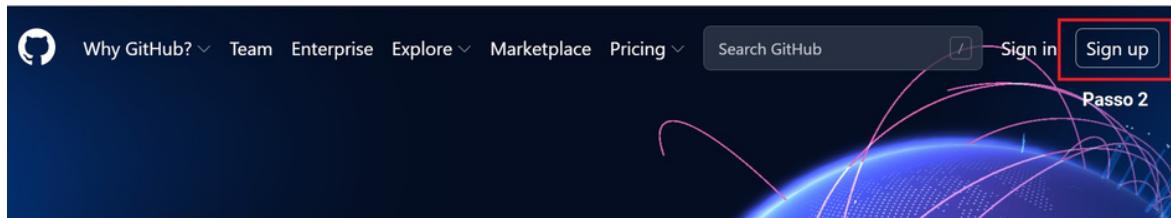
Passo 1 - Dentro do buscador do Google, vamos digitar github e enter para confirmar. E logo em seguida vamos clicar no primeiro link da busca - <https://github.com>



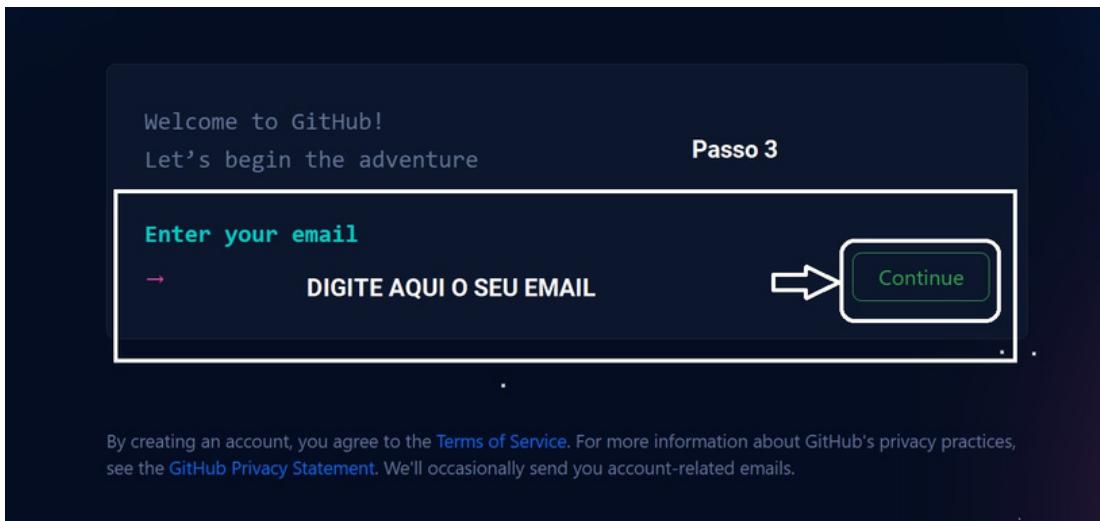
/igor-rebolla

# Instalação do GitHub

Passo 2 - Ao clicarmos em <https://github.com> – o site do github será aberto e vamos clicar em Sign up



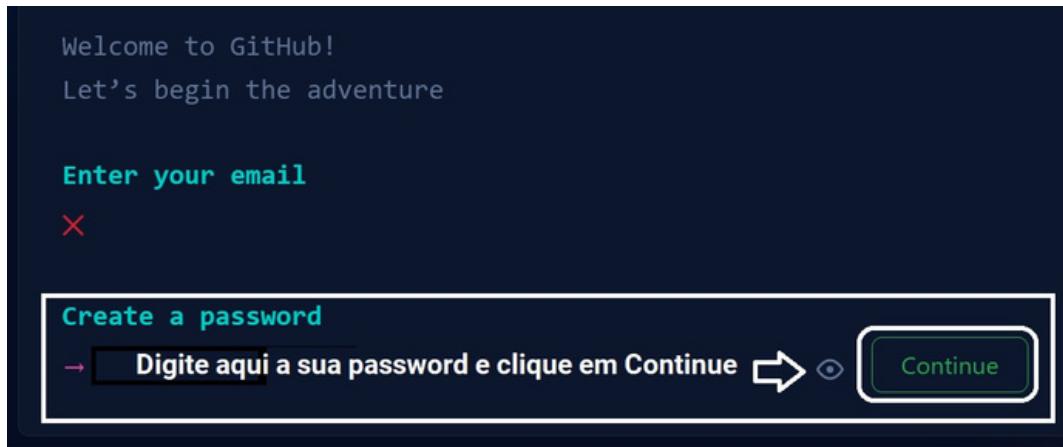
Passo 3 - Vamos digitar um email válido e clicaremos em Continue:



/igor-rebolla

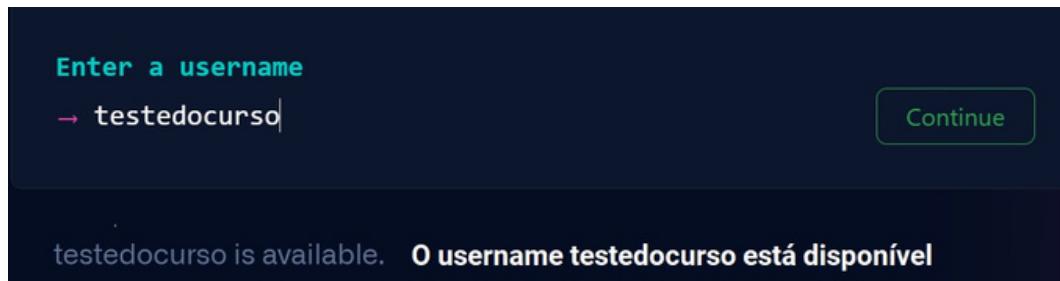
# Instalação do GitHub

Passo 4 - Vamos digitar uma senha (password) válida e clicaremos em Continue



Passo 5 - Vamos digitar um username válido para sermos identificados dentro do GitHub e clicaremos em Continue.

Aqui eu escolhi o username (testedocurso) e ele está disponível(available).



# Instalação do GitHub

Passo 6 - Aqui ele está perguntando "Se gostaríamos de receber atualizações e anúncios por e-mail?" - Eu optei por Não (Digitei a letra n) e cliquei em Continue.



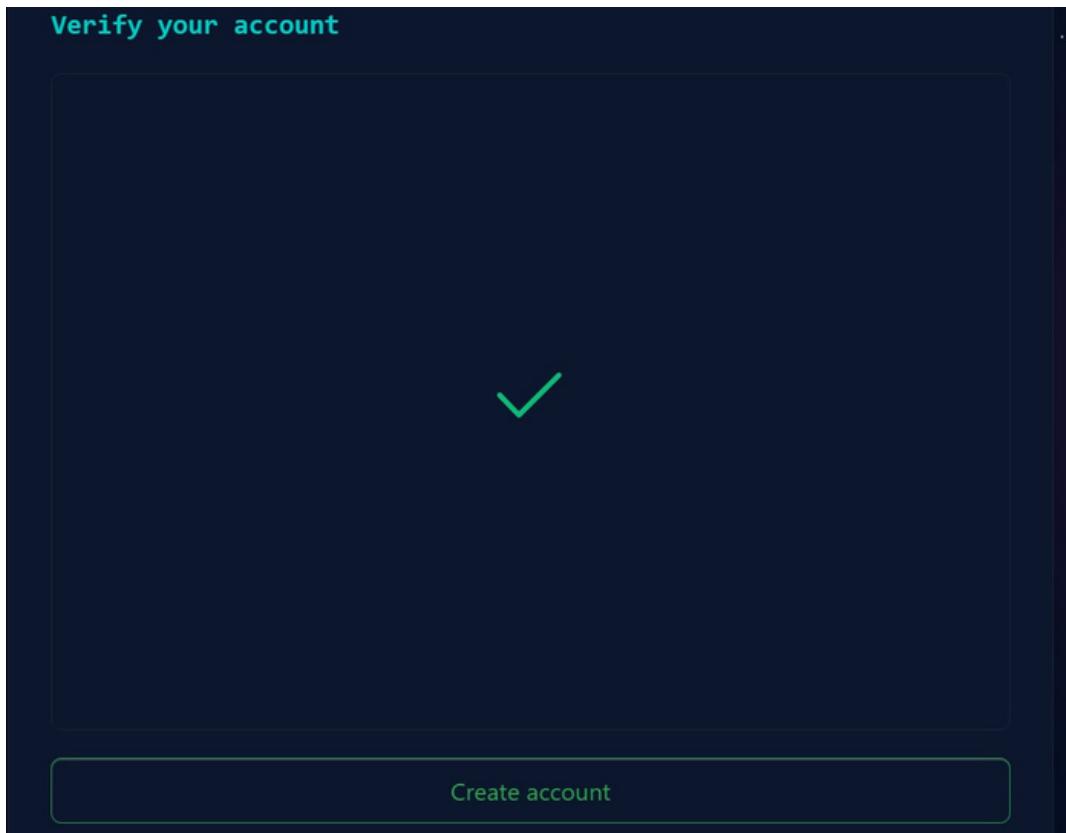
Passo 7 - Aqui ele vai pedir para você resolver um desafio (Escolha a galáxia espiral) - 5 imagens da galáxia espiral apareceram aqui para que eu resolvesse.



/igor-rebolla

# Instalação do GitHub

Passo 8 – Depois que as 5 imagens foram resolvidas, aparece um: Selo de validação verde no meio da tela e o botão Create account fica habilitado e é nesse botão que iremos clicar para criarmos nossa conta com as informações passadas até o momento dentro do GitHub.



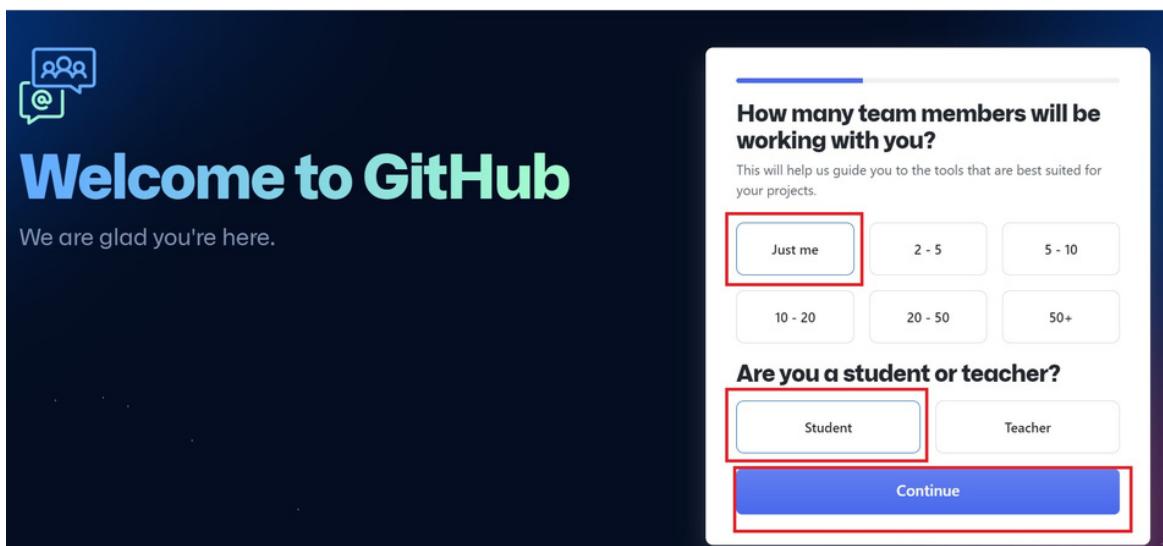
# Instalação do GitHub

Passo 9 – O GitHub enviou um código de segurança para o email cadastrado no passo 3 da sua instalação. Basta ir até esse email pegar o código e inseri-lo aqui. Peço desculpas, não dei o print da tela nesse passo.

Ao digitar os 8 dígitos do código ele automaticamente vai para a tela com 2 perguntas:

- Quantas membros do time vão trabalhar contigo?
- Você é um estudante ou professor?

Vou selecionar (Just me) e (Student) e clicarei em Continue

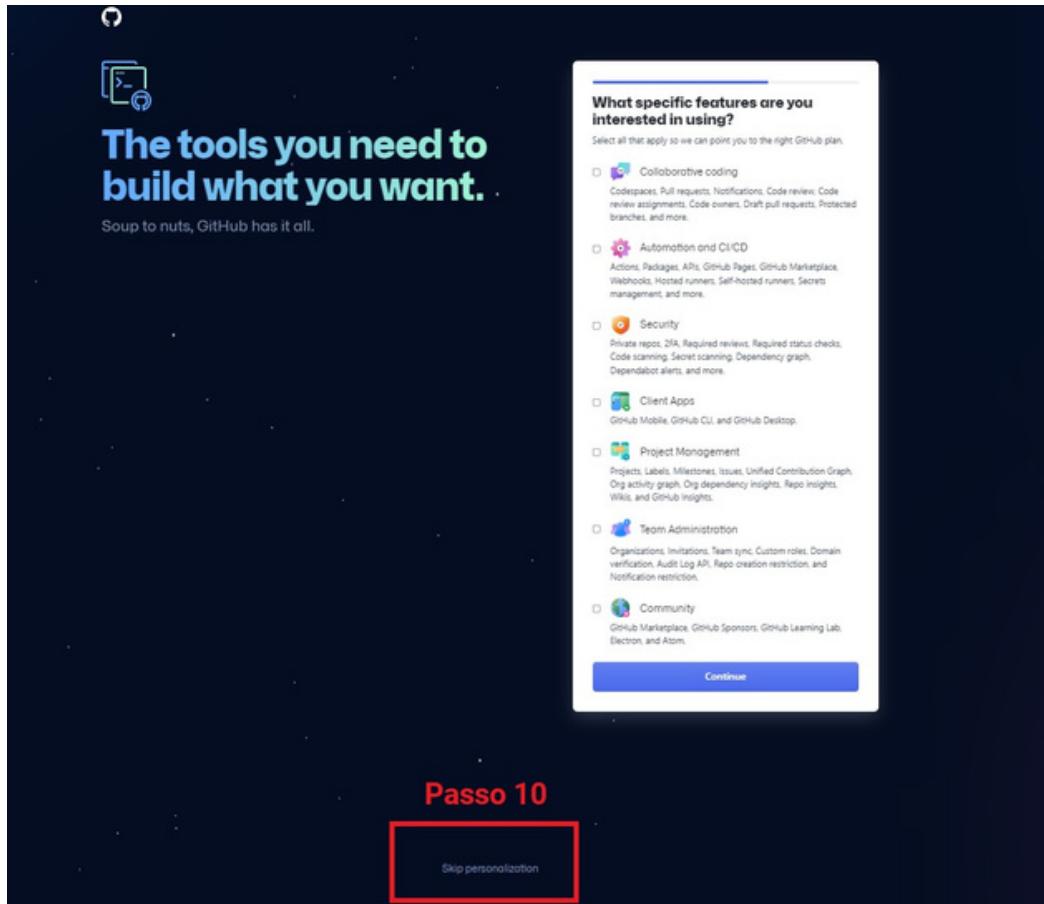


/igor-rebolla

# Instalação do GitHub

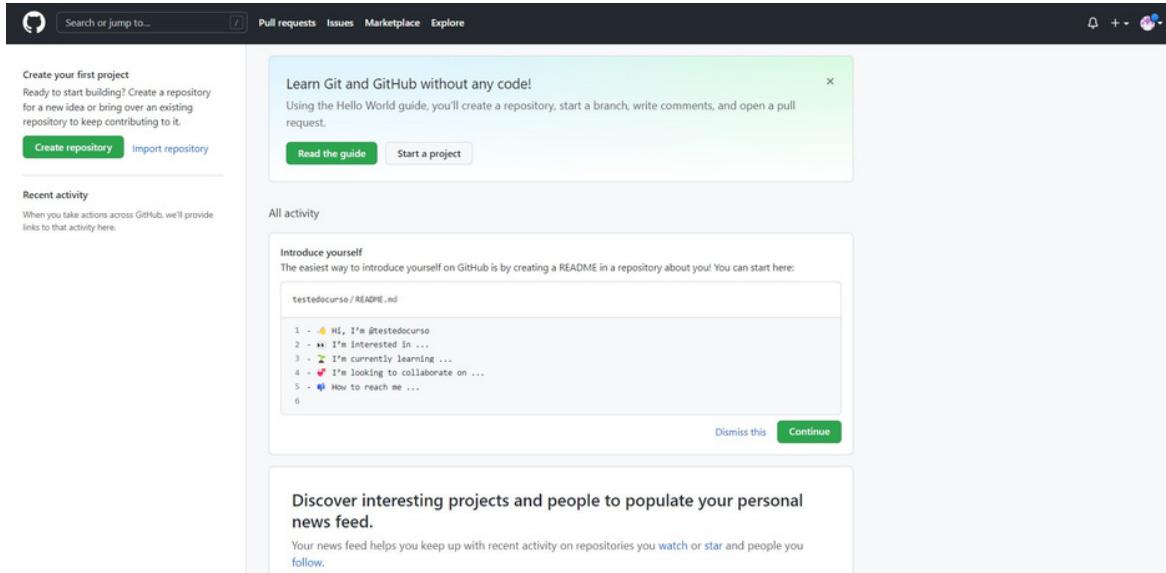
Passo 10 - Aqui ele está perguntando: "Quais features específicas você está interessado em usar?"

Eu vou clicar em Skip Personalization (Pular personalização)



# Instalação do GitHub

Passo 11 – Deu tudo certo com a nossa instalação. Agora já temos o nosso Dashboard do GitHub criado.

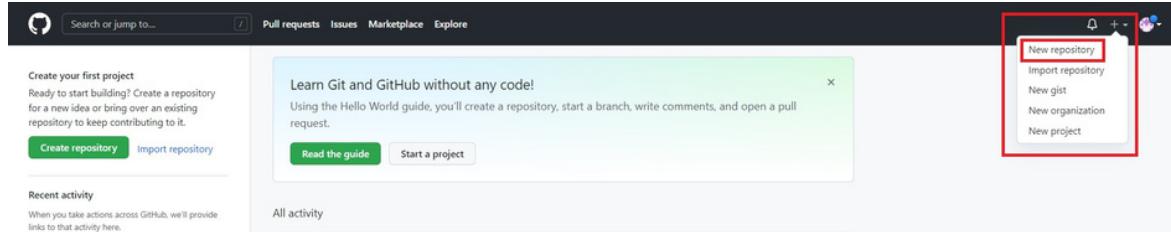


/igor-rebolla

# Exemplo Prático do GitHub

Passo 12 - Agora que o GitHub já está instalado, vamos criar o nosso primeiro repositório (para entender o seu funcionamento).

- Clicar no sinal de + no canto superior direito da tela;
- Clicar em New Repository (Novo Repositório)



/igor-rebolla

# Exemplo Prático do GitHub

Passo 13 – Aqui vamos definir o nome do nosso repositório. Notem que antes de definirmos o nome... em Owner (ele vem o nome do usuário) esse foi o nome o qual configuramos em uma das etapas da instalação do GitHub.

- Eu vou definir o nome desse repositório como Curso-FrontEnd;
- Aqui vamos deixar como Public (público) o acesso a esse repositório;
- Vou criar em Create repository

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \* Repository name \*

testedocurso  /  Curso-FrontEnd  Nome do Repositório

Great repository names are short and memorable. Need inspiration? How about refactored-train?

Description (optional)

**Public** Anyone on the internet can see this repository. You choose who can commit. **Vamos deixar como Public mesmo.**

**Private** You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore Choose which files not to track from a list of templates. [Learn more.](#)

Choose a license A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

E vamos clicar em Create Repository



/igor-rebolla

# Exemplo Prático do GitHub

Passo 14 - Aqui ele mostra o nosso repositório criado e alguns comandos que já foram utilizados quando vimos o GIT (nessa mesma aula).

## **git init**

iniciamos um repositório vazio(empty)

## **git add README.md**

aqui a diferença é que ele está adicionando apenas o README.md e no nosso exemplo usamos o git add . (pois adicionamos todos os arquivos disponíveis - index.html e o style.css)

## **git commit -m "first commit"**

fizemos o nosso primeiro commit

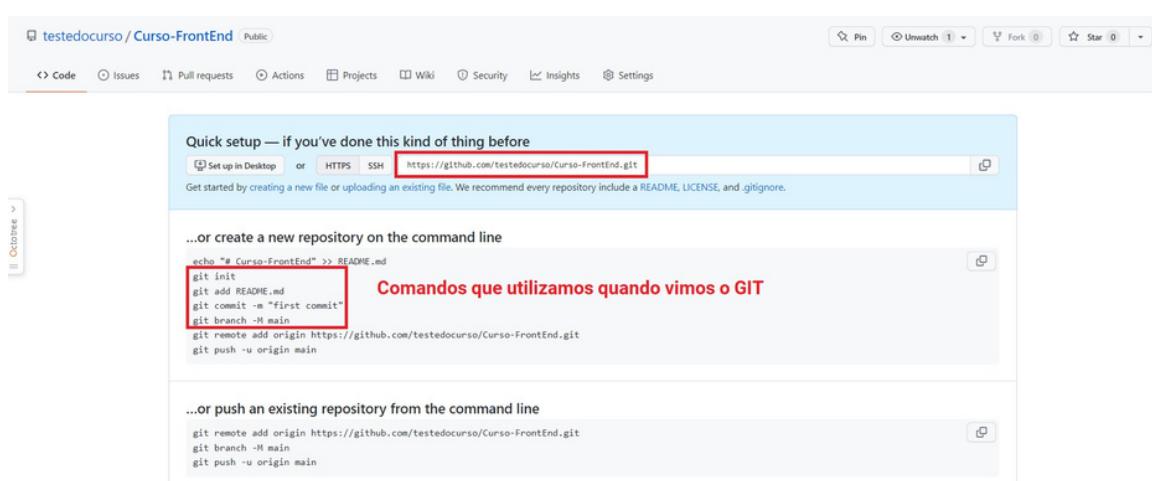
## **git branch -M main**

aqui mudamos o nome da nossa branch de master para main

Ainda não vimos esses 2 comandos abaixo:

git remote add origin https://github.com/testedocurso/Curso-FrontEnd.git

git push -u origin main



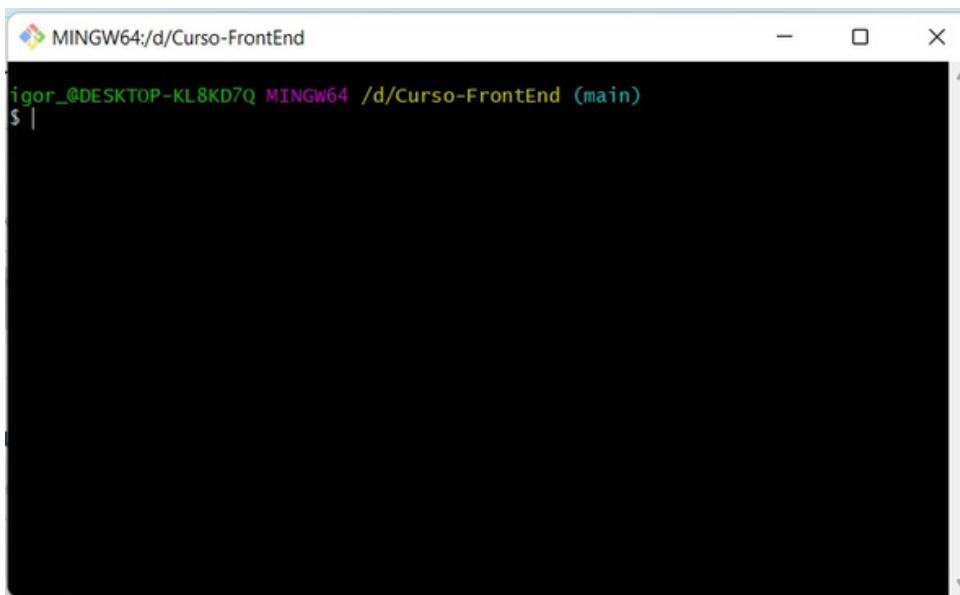
/igor-rebolla

# Exemplo Prático do GitHub

Passo 15 – Vamos abrir o Git Bash, só relembrando os passos:

1. Localizar onde a pasta Curso-FrontEnd foi salva dentro do computador;
2. Clicar 2x para abrir essa pasta;
3. Clicar com o botão direito do mouse e escolher a opção Git Bash Here

Essa é a tela que será mostrada (Já com o main) alterado no lugar do master:



```
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (main)  
$ |
```



# Exemplo Prático do GitHub

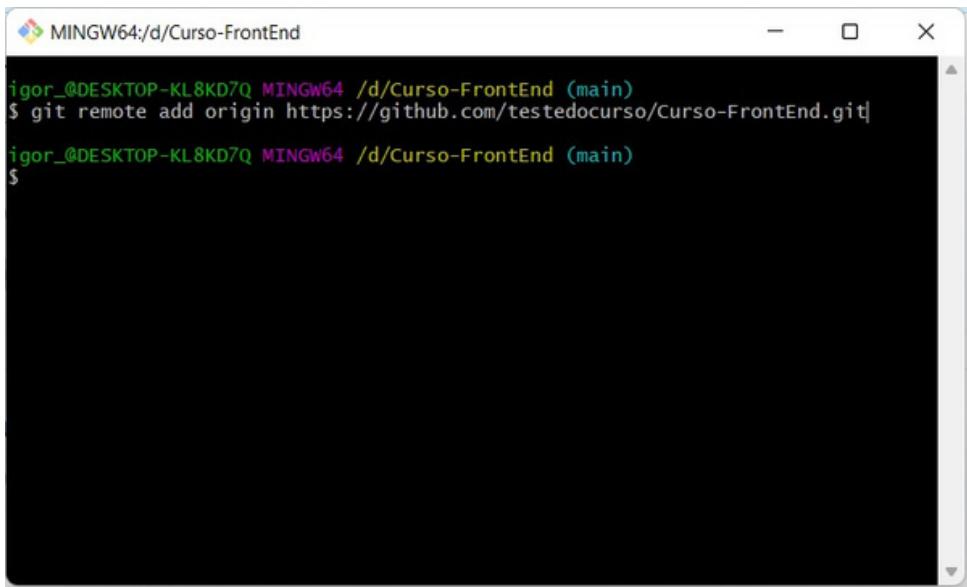
Passo 16 – Com a tela do Git Bash aberta vamos seguir com os 2 comandos que ficaram faltando fazer quando vimos na parte do Git. E o primeiro desses comandos será o:

```
git remote add origin https://github.com/testedocurso/Curso-FrontEnd.git
```

Notem que ao pressionarmos o enter parece que nada aconteceu.

E qual é a função desse comando?

É criar a conexão entre o repositório local(GIT) que está em nosso computador com o repositório remoto(GitHub).



```
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (main)
$ git remote add origin https://github.com/testedocurso/Curso-FrontEnd.git|
```

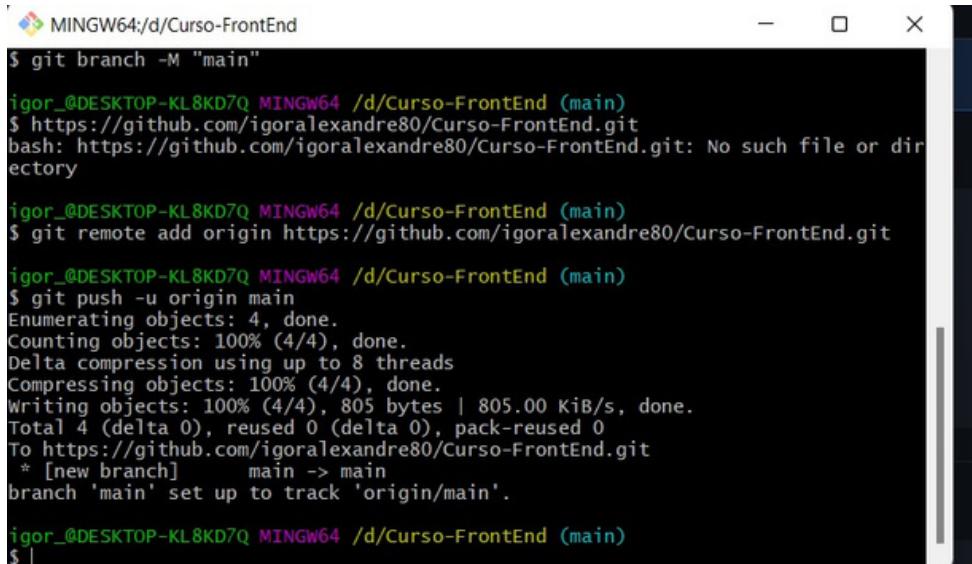


# Exemplo Prático do GitHub

Passo 17 – Só que ainda não enviamos nenhum arquivo para o nosso repositório remoto (github). Para enviarmos esses arquivos vamos usar o comando:

```
git push -u origin main
```

Esse comando praticamente empurra(push) os commits que estão no nosso repositório local(GIT) para o repositório remoto(GitHub).



```
MINGW64:/d/Curso-FrontEnd
$ git branch -M "main"
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (main)
$ https://github.com/igoralexandre80/Curso-FrontEnd.git
bash: https://github.com/igoralexandre80/Curso-FrontEnd.git: No such file or directory

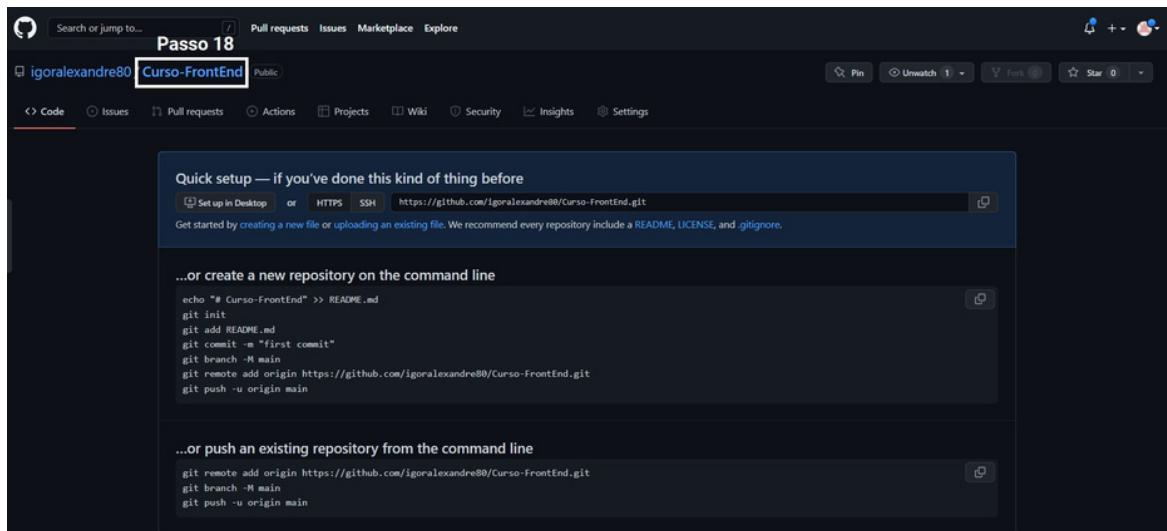
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (main)
$ git remote add origin https://github.com/igoralexandre80/Curso-FrontEnd.git
igor_@DESKTOP-KL8KD7Q MINGW64 /d/Curso-FrontEnd (main)
$ git push -u origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 805 bytes | 805.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/igoralexandre80/Curso-FrontEnd.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```



# Exemplo Prático do GitHub

Passo 18 – Arquivos enviados com sucesso do repositório local para o repositório remoto. Vamos visualizar como esses arquivos ficaram dentro do GitHub.

Para isso só clicar em Curso-FrontEnd no canto superior esquerdo da tela conforme imagem abaixo:



# Exemplo Prático do GitHub

Passo 19 – Podemos verificar que a pasta Curso-FrontEnd foi adicionada no repositório remoto (Github) com seus 2 arquivos (index.html e o style.css). Isso permite:

- Acessar remotamente;
- Informar o portfólio para o recrutador;
- Compartilhar os arquivos com o cliente;
- Trabalhar com mais pessoas no mesmo projeto;
- Etc...

Basta acessar por esse endereço aqui:

<https://github.com/igoralexandre80/Curso-FrontEnd>

The screenshot shows a GitHub repository page for 'igoralexandre80/Curso-FrontEnd'. The repository is public and contains one branch ('main') and one commit. The commit was made by 'igoralexandre80' 13 minutes ago, with the message 'Esse é o meu primeiro commit'. The commit includes two files: 'index.html' and 'style.css', both with the same commit message. The repository has 1 branch and 0 tags. There are buttons for 'Go to file', 'Add file', and 'Code'.



/igor-rebolla

# **Sugestões de prática para essa aula:**

1. Instalar o Git (Utilizando o passo a passo dessa aula);
2. Instalar o GitHub (Utilizando o passo a passo dessa aula);
3. Criar o seu primeiro repositório no Github e passar os arquivos do repositório local (GIT) para o repositório remoto (GitHub)

Programação é prática, curiosidade e repetição!!!!

