

4º e-book

Curso: Front-End do Bem



100% Gratuito | Baixou... Estudou | Sem pedir e-mail

- O que é o JavaScript(JS) ?
 - Var, Let e Const
 - Numbers e Strings
 - Template Strings
 - Null e Undefined
 - Array(s)
- Sugestões de Prática no final de cada tema abordado
 - E muito mais...



/igor-rebolla

Conteúdo:

3º Pilar do Curso FrontEnd do Bem (JavaScript(JS))

- O que é o JavaScript(JS) ?
- Como configurar o nosso ambiente
- Instalando uma Extensão
- Conceito de Servidor(Server)
- Colocando o Live Server no Ar
- Sugestões de Prática

- Como adicionar o JavaScript(JS) em uma Página Web
- Exemplos
- O Console do Navegador
- Exemplos
- Sugestões de Prática

- O que é essa Tal de Variável ?
- let
- const
- var
- 3 Regrinhas para nomearmos variáveis
- Sugestões de Prática



Conteúdo:

3º Pilar do Curso FrontEnd do Bem (JavaScript(JS))

- Uma Visão Geral sobre Tipo de Dados
- Comentários
- Exemplos de Comentários
- Numbers
- Exemplos de Numbers
- Sugestões de Prática

- Numbers (Parte Final)
- Exemplos de Numbers (Parte Final)
- Strings
- Exemplos de Strings
- Sugestões de Prática

- Concatenação de Strings (+1 exemplo)
- Template Strings
- Exemplo de Template Strings
- Template HTML
- Exemplo de Template HTML
- Sugestões de Prática



Conteúdo:

3º Pilar do Curso FrontEnd do Bem (JavaScript(JS))

- Null
- Exemplos de Null
- Undefined
- Exemplos de Undefined
- Null e Undefined (uma duplinha do barulho - Revisão)
- Sugestões de Prática

- Array(s)
- Exemplos de Array(s)
- Array é Pop, Array é Tech (Método Pop)
- Exemplo do Método Pop
- Array é Push, Array é Tech (Método Push)
- Exemplo do Método Push
- Sugestões de Prática

- Booleans
- Exemplos de Booleans
- Método Includes()
- Exemplos do Método Includes()
- Sugestões de Prática



Conteúdo:

3º Pilar do Curso (JavaScript(JS))

- Operadores de Comparação
- 16 Exemplos de Operadores de Comparação (Strings e Numbers)
- Sugestões de Prática
- Pergunta do Amigo Internauta: Igor de São Paulo - SP



O que é o JavaScript (JS) ?

É uma linguagem de programação que permite a você criar conteúdo que se atualiza dinamicamente, controlar vídeos, imagens animadas, e etc...
Fonte (MDN)

De forma lúdica:

O que eu entendo da definição Acima (Já acrescentando os 2 pilares vistos em nossas aulas anteriores: HTML e CSS):

O HTML é responsável por criar a estrutura(paredes, interruptores de luz) da casa, o CSS é responsável por colorir essas paredes e definir o formato do interruptor e o JavaScript é o responsável em dar movimento ao abrir e fechar uma passagem secreta em uma dessas paredes e a ligar e desligar o interruptor de luz.



Como Configurar o Nosso Ambiente

Antes de começarmos a escrever o nosso código em JavaScript é necessário que se escolha um editor de texto, como nós já utilizamos o VS CODE em nossas aulas anteriores e no projeto Pet Shop do Bem (Segundo E-book), iremos utiliza-lo por aqui também

Se você está chegando agora nesse material e não sabe como baixar o VS CODE, vou deixar o link aqui abaixo da aula que faz referência a isso (Está passo a passo e bem detalhado de como fazer).

https://www.linkedin.com/posts/igor-rebolla_aula01-activity-6874671504194318337-HFs0/?utm_source=share&utm_medium=member_desktop

Obs.: Caso você tenha algum problema com o Link Acima é só ir no meu perfil do Linkedin, ir na seção Destaques que todas as 14 aulas anteriores a essa + os 2 e-books vão estar lá... separadas para facilitar o Vosso entendimento e ir estudando no seu ritmo e conforme sua necessidade. Ah, já ia me esquecendo (tanto as aulas quanto os e-books) estão disponíveis para download e/ou impressão (Tudo gratuito).

Agora se você já tem o VS CODE instalado, bora ver como instalamos uma Extensão(Extensions) que vai facilitar bastante o nosso desenvolvimento.



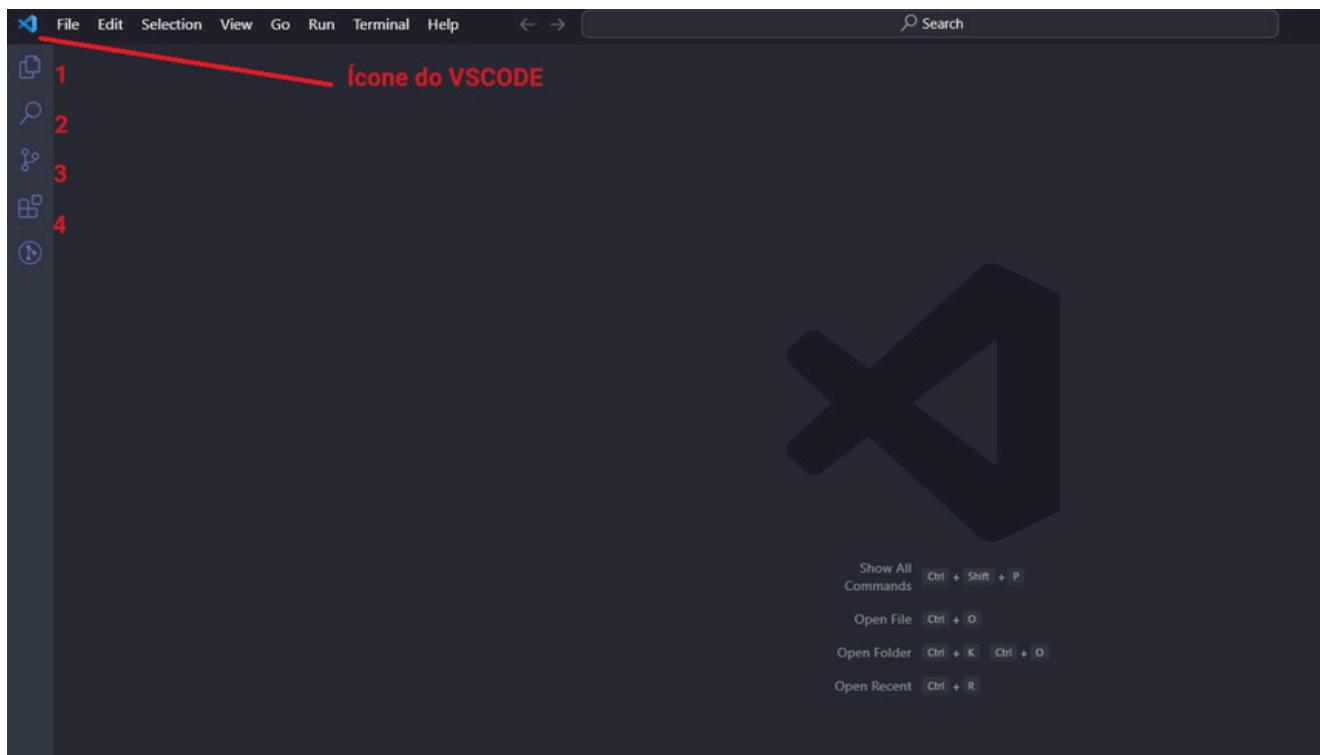
Instalando a Extensão Live Server

O Live Server permite que seja visualizado no navegador(Google Chrome) os exemplos que iremos construir durante a nossa jornada aqui no curso.

Igor, e como eu instalo esse tal de Live Server?

Simbora mostrar...

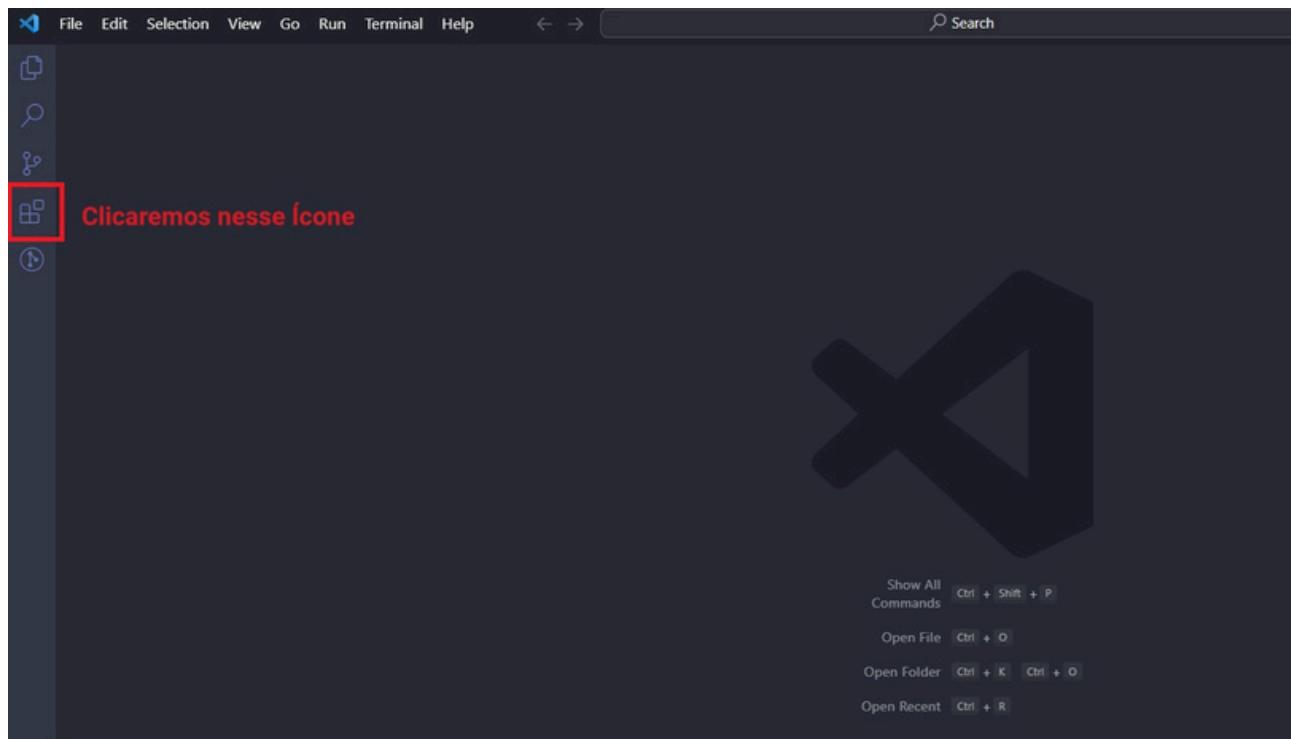
Dentro do VS CODE, logo abaixo do ícone do VS CODE tem uma barra na lateral esquerda onde podemos ver alguns ícones:



Instalando a Extensão Live Server

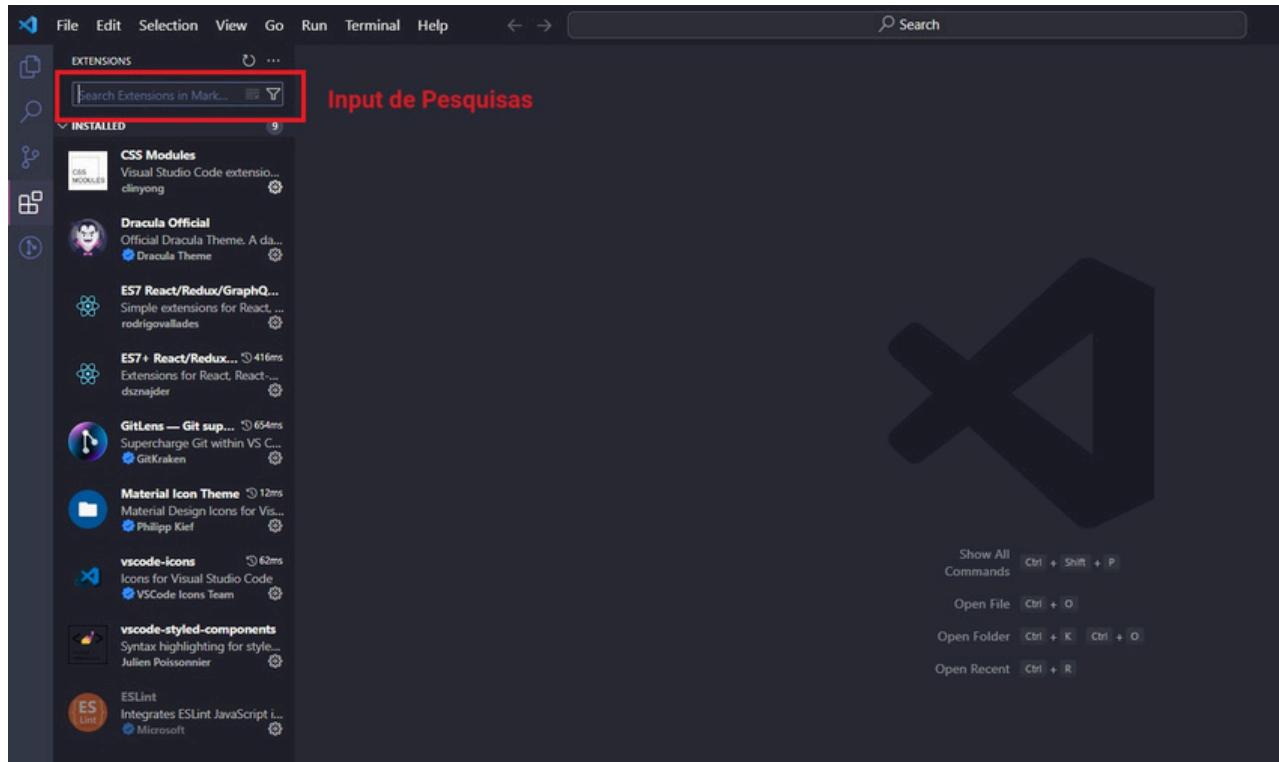
- 1 - O primeiro ícone são duas páginas quase que intercaladas (Não queremos você agora - Parte 1)
- 2 - O segundo ícone é uma lupa (Não queremos você agora - Parte 2)
- 3 - O terceiro ícone parece aquela cena final do Velozes e Furiosos onde cada um segue o seu caminho (Não queremos você agora - Parte 3)
- 4 - O quarto ícone são 3 quadradinhos juntos e um quarto quadradinho separado (AEEE UHUUU CHEGAMOS ONDE QUERIAMOS).. Esse carinha aqui é o Extensão(Extensions)!!!

Clicaremos no nosso amigo Extensions



Instalando a Extensão Live Server

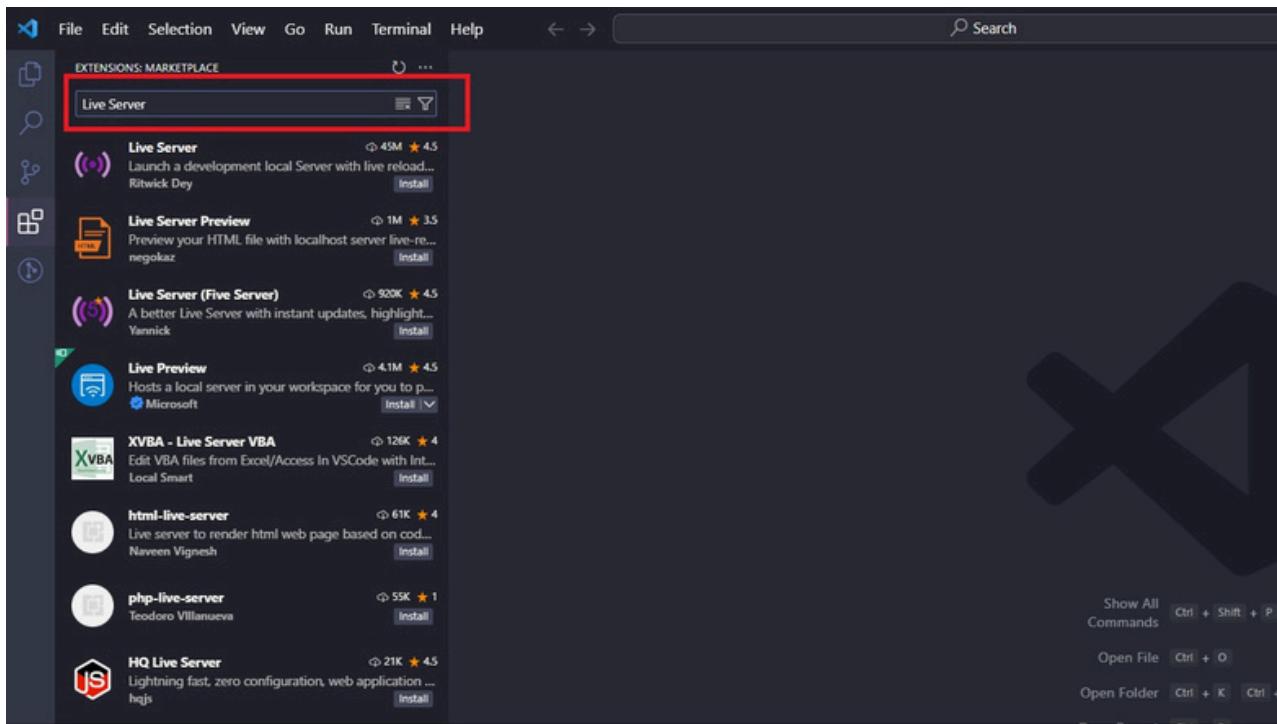
Ao clicarmos será aberto na parte superior um campo de input para pesquisarmos a extensão a qual desejamos instalar



Vamos digitar Live Server e pressionar o enter para buscar por essa extensão



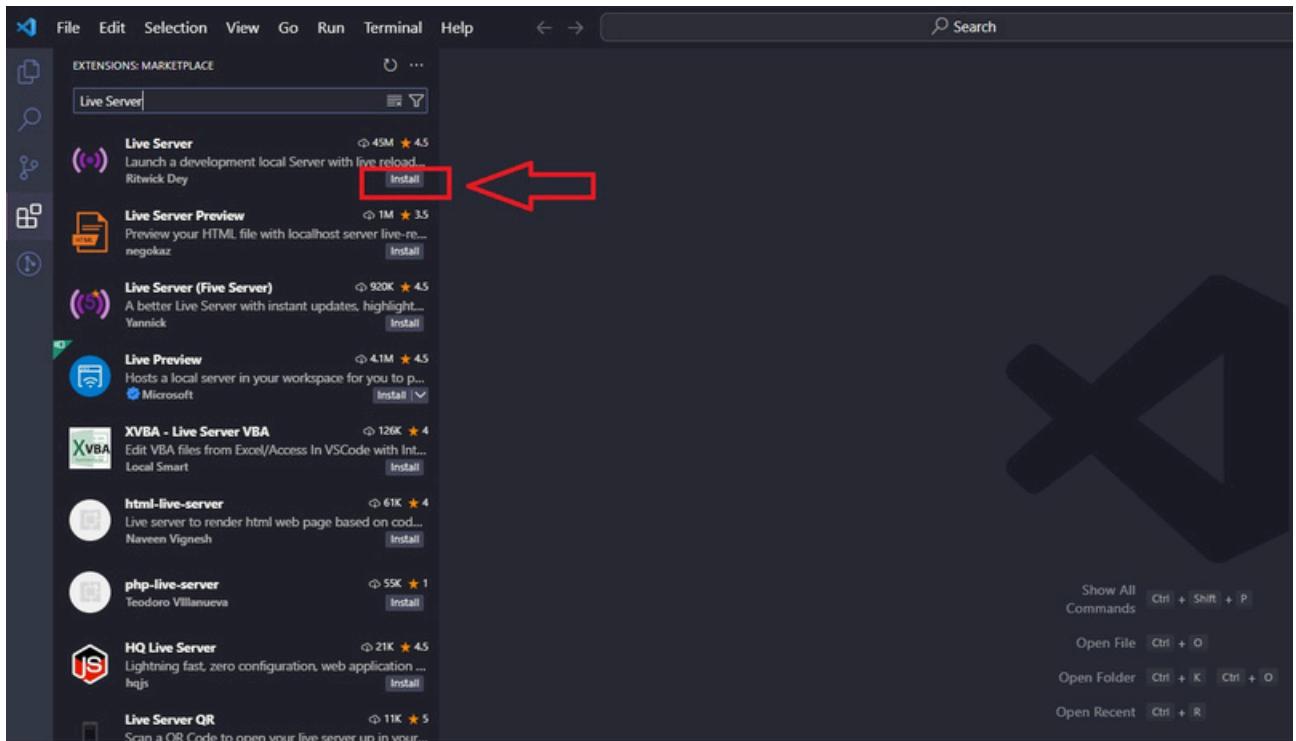
Instalando a Extensão Live Server



Vamos instala-la clicando em Install



Instalando a Extensão Live Server



Agora que concluímos a instalação do Live Server precisamos entender outro conceito IMPORTANTESSIMO sobre SERVIDOR.



O que é esse tal de Servidor(Server)

Igor, estavamos instalando o Live Server e agora você vem falar sobre Servidor.. Tá e o que vem a ser esse tal de Servidor?

Calma que daqui a pouco você vai entender a ligação entre o Live Server e o Servidor

Servidor: Quando nós abrimos um navegador(Chrome) e digitamos um endereço, por exemplo: <http://sitedoigor.com.br> dentro do campo de pesquisa e ao pressionarmos o enter(nesse exato momento que o enter é apertado) uma REQUISIÇÃO(REQUEST) é enviada para outro Computador que fica lá longe e que tem todas as informações do Site que queremos acessar, o nome desse carinha é o SERVIDOR WEB(WEB SERVER)

Igor, pode resumir melhor isso?

Posso sim...

1 - No momento em que digitamos <http://sitedoigor.com.br> e pressionamos o enter enviamos uma Requisição(Request) para um SERVIDOR WEB(WEB SERVER) que recebe essa Requisição(Request) e fala... "Hum... o que temos por aqui!!!"

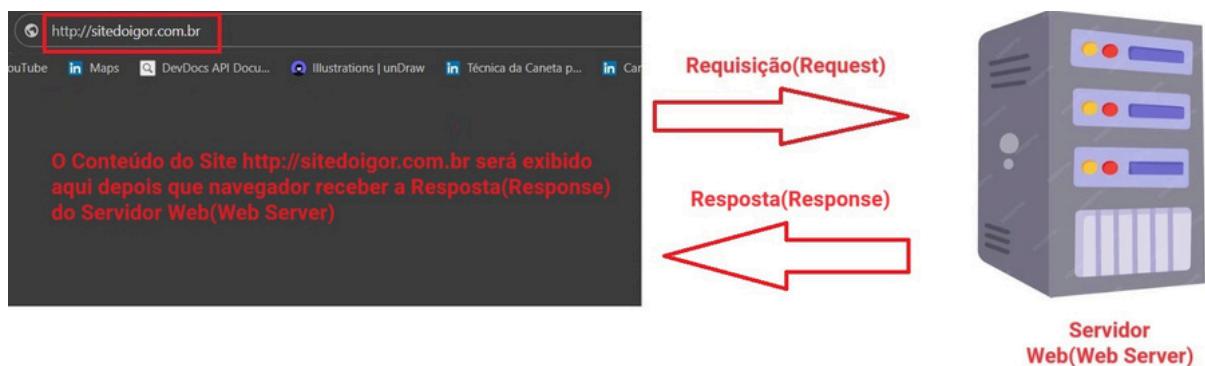


O que é esse tal de Servidor(Server)



2 - O Servidor Web(Web Server) entende que acabou de receber uma Requisição(Request) e fala beleza pura amigão estou te enviando uma Resposta(Response) para esse Navegador (Chrome). e essa RESPOSTA(RESPONSE) possui dentro dela (O HTML, O CSS, O JAVASCRIPT, OS VIDEOS, AS IMAGENS) e os demais itens que fazem parte desse site.

O Navegador(Chrome) recebe essa RESPOSTA(RESPONSE) e só após é que será carregado o Site: <http://sitedoigor.com.br>



Servidor(Server) e a Extensão(Extension) Live Server

Agora que vem a parte que vai ser esclarecida por que paramos com a parte do Live Server para explicarmos sobre o que é um Servidor.

Faremos algumas atividades durante as nossas aulas e nessas atividades teremos alguns arquivos e esses arquivos serão armazenados dentro do seu computador(Local) ao invés de outro computador como fizemos no exemplo para acessar o site do Igor (onde explicamos sobre request e response).

Já que não temos esse outro computador, precisamos dar um jeito maroto para que o nosso computador venha a servir esses arquivos ao Navegador(Chrome) como se estivéssemos acessando um Site real.

E para fazermos isso vamos utilizar um carinha chamado SERVIDOR LOCAL(LOCAL SERVER) e dessa forma o nosso computador vai funcionar como um Servidor(Server) para as nossas atividades aqui do curso.



Servidor(Server) e a Extensão(Extension) Live Server

Usando o Servidor Local(Local Server) é possível especificar um endereço local na busca do Navegador(Chrome), e ao pressionarmos o enter vamos dizer ao nosso computador: "Computador, por gentileza poderia servir os arquivos HTML, O CSS, O JAVASCRIPT, OS VIDEOS, AS IMAGENS e os demais itens que fazem parte do site) para o Navegador(Chrome)" o qual estamos utilizando, para que o conteúdo do site seja visualizado.

O endereço que vamos usar, tem o nobre nome de LOCALHOST que mostra que estamos usando Servidor Local(Local Server) no nosso computador. E essa é a função da EXTENSÃO LIVE SERVER que instalamos dentro do VS CODE, por isso a importância de entendermos o conceito de Servidor(Server).

Ufa, consegui juntar os paranaues, estava ficando preocupado!!!

Igor, agora eu entendi o que é um Servidor e a função da EXTENSÃO LIVE SERVER que instalamos dentro do VS CODE e quando vamos ver como utilizar o Live Server (quero ver esse trem funcionando) ?

Que rufem os tambores...

Chegou o grande momento: Como vamos colocar o servidor local para funcionar usando o Live Server.



/igor-rebolla

Colocando a Extensão Live Server no Ar(Pra funcionar)

Precisamos antes de tudo, criar uma nova pasta (local onde os arquivos) do nosso curso vão ficar alocados e assim poderemos deixar nossos arquivos de forma organizada dentro do VSCode.

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

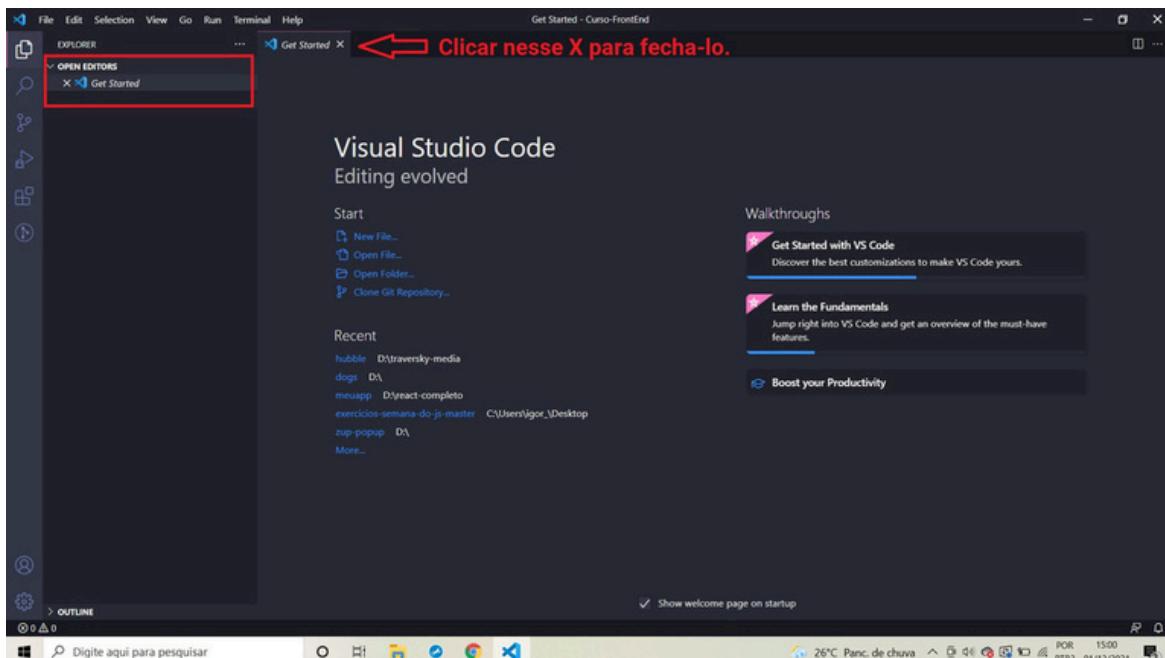
- Abrir o Explorador de Arquivos ou Windows Explorer;
- Selecionar o Local onde a pasta do curso será criada (eu escolhi o D:/);
- Clicar com o botão direito do mouse e escolher a opção (novo e depois pasta);
- Digitar o nome da Pasta – Aqui eu optei por Curso-FrontEnd

Obs.: Para quem está acompanhando as 14 aulas anteriores a essa aqui, a pasta Curso-FrontEnd já foi criada. Esse procedimento de criar a pasta é para quem começou a acompanhar nossas aulas agora.



Colocando a Extensão Live Server no Ar(Pra funcionar)

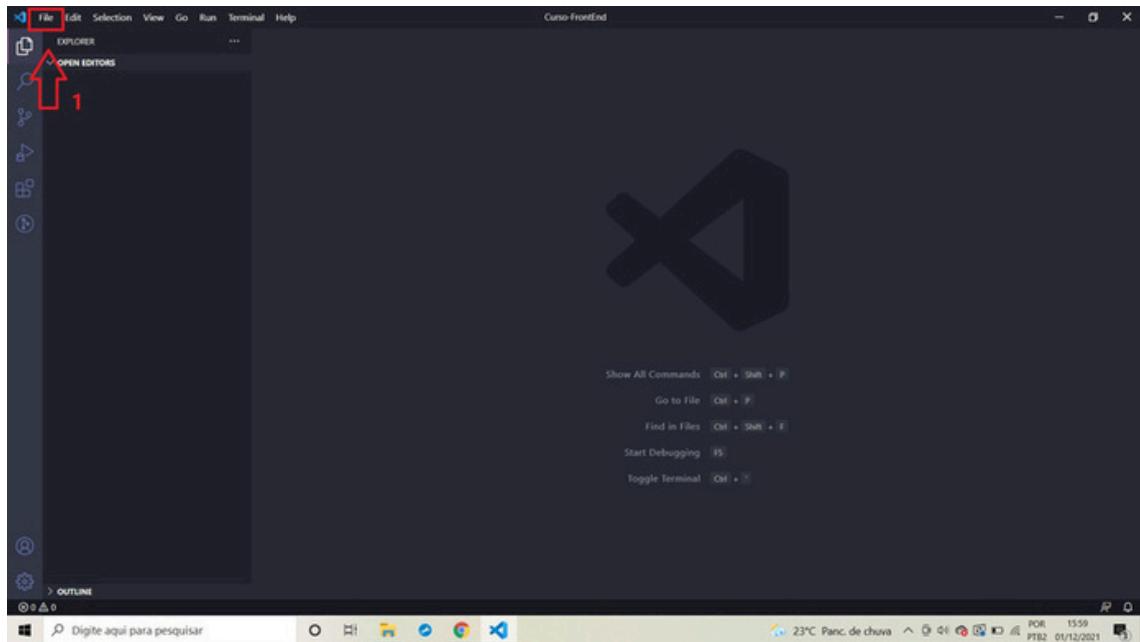
Pasta Curso-FrontEnd devidamente criada, vamos abri-la dentro do VSCode. Antes vamos fechar o Get Started (para fazer isso basta clicar no X) conforme figura abaixo:



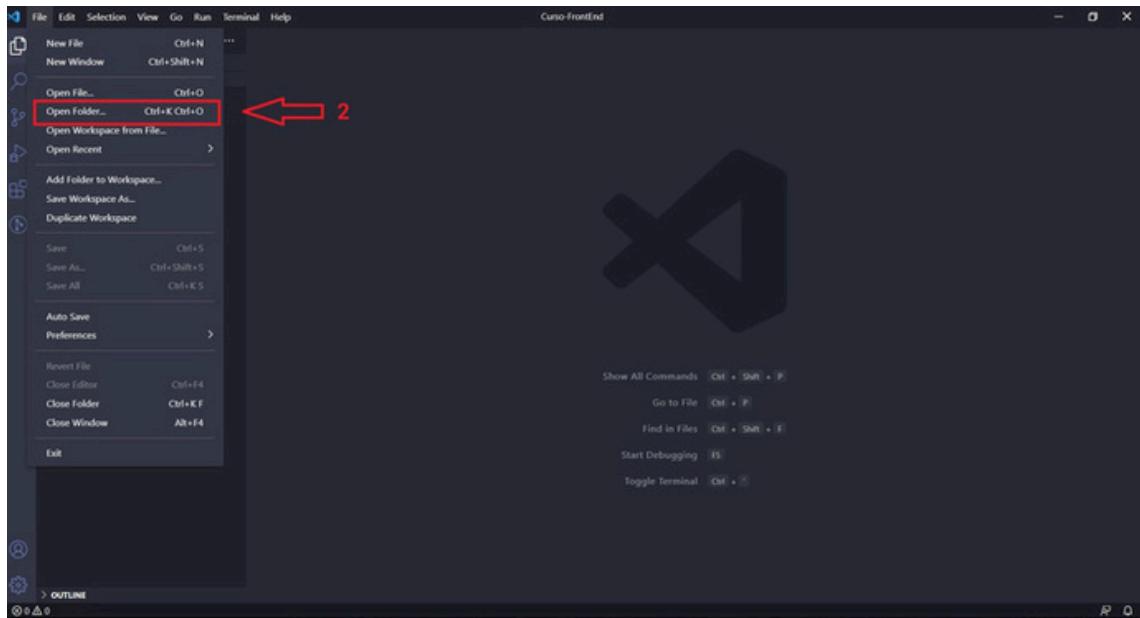
Bora abrir dentro do VSCode a pasta Curso-Front End. Para fazer isso, clicar em FILE (Passo 1)



Colocando a Extensão Live Server no Ar(Pra funcionar)



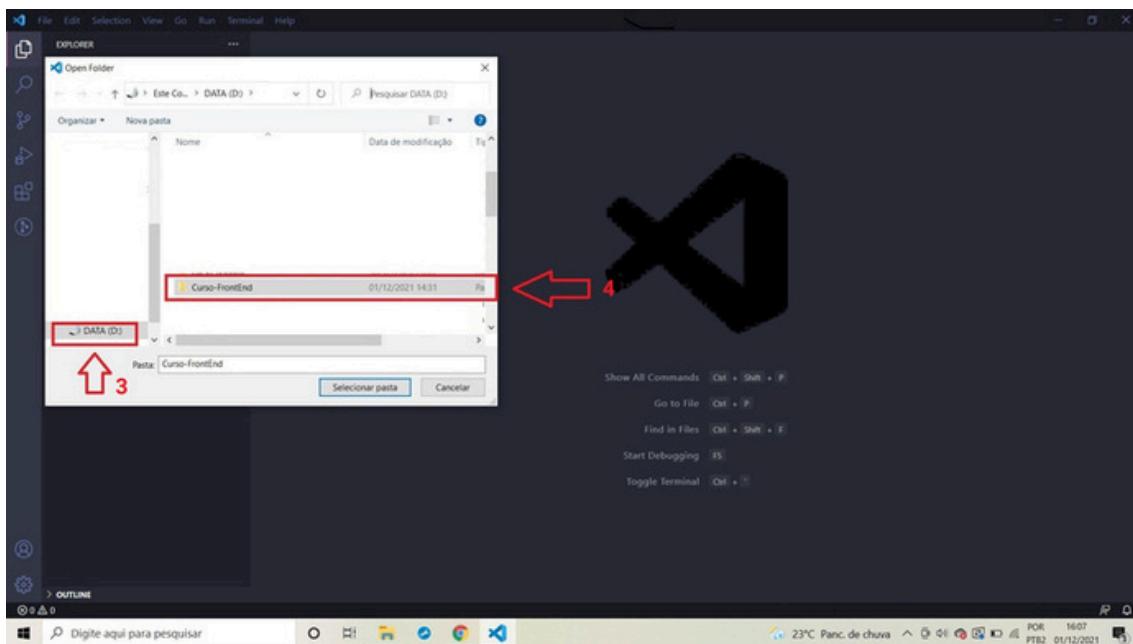
E clicaremos em Open Folder (Passo 2)



/igor-rebolla

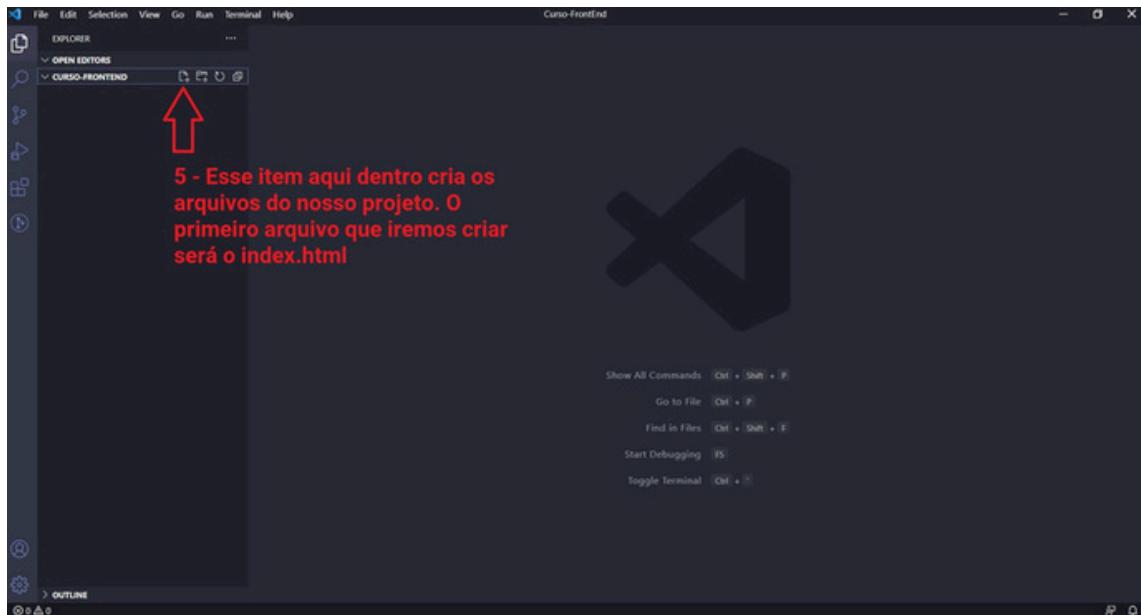
Colocando a Extensão Live Server no Ar(Pra funcionar)

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 VSCode.

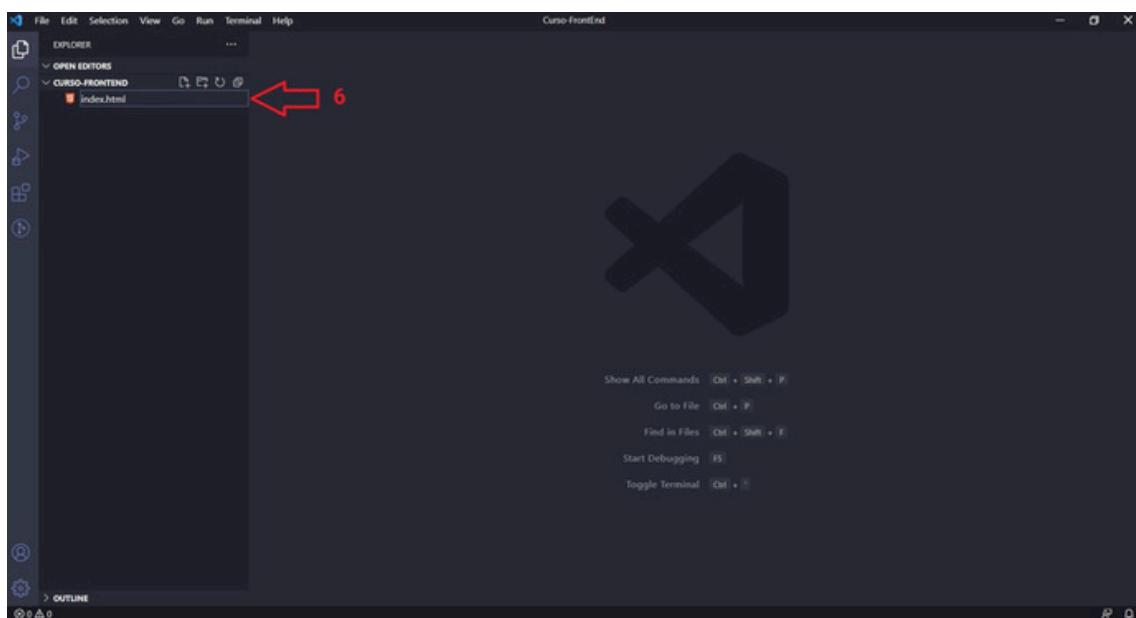


Pronto a Pasta Curso-FrontEnd está aberta do lado esquerdo da tela (o lado esquerdo é o local onde o VSCode disponibiliza toda a estrutura de pastas e arquivos que iremos utilizar em nosso projeto. Notem que não tem nada embaixo da pasta Curso-FrontEnd nesse momento. Vamos clicar no desenho do arquivo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

Colocando a Extensão Live Server no Ar(Pra funcionar)

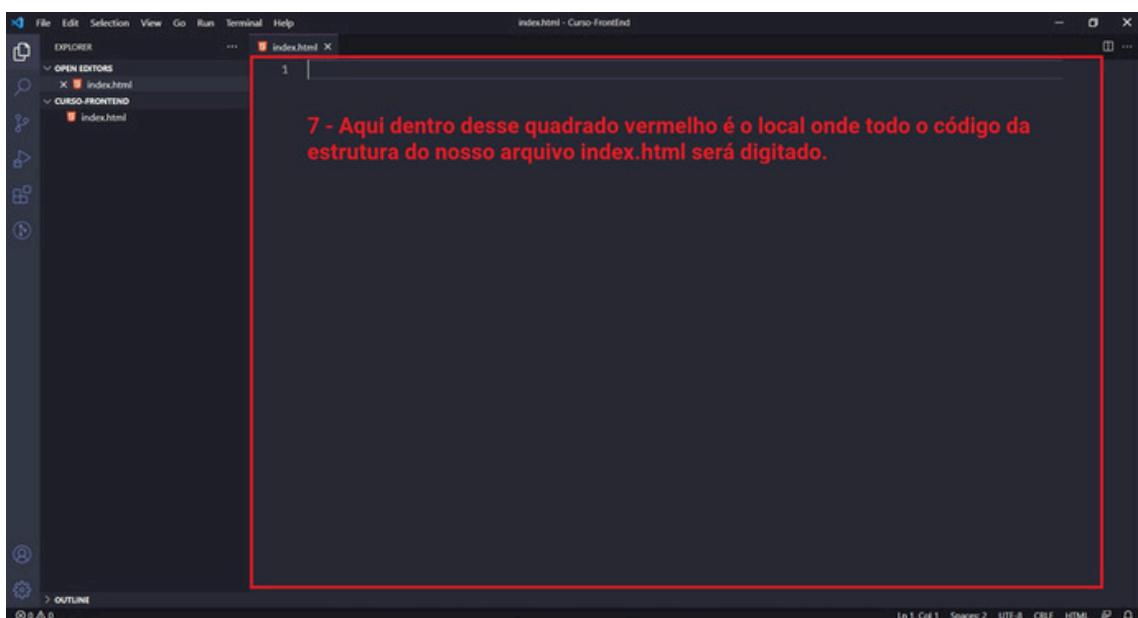


Vamos nomear o primeiro arquivo do nosso curso, ao clicarmos no desenho do arquivo logo abaixo da pasta Curso-FrontEnd o cursor ficará piscando para ser digitado algo é aqui que daremos o nome de: index.html e logo em seguida pressionaremos o enter lx para gravar esse nome.



Colocando a Extensão Live Server no Ar(Pra funcionar)

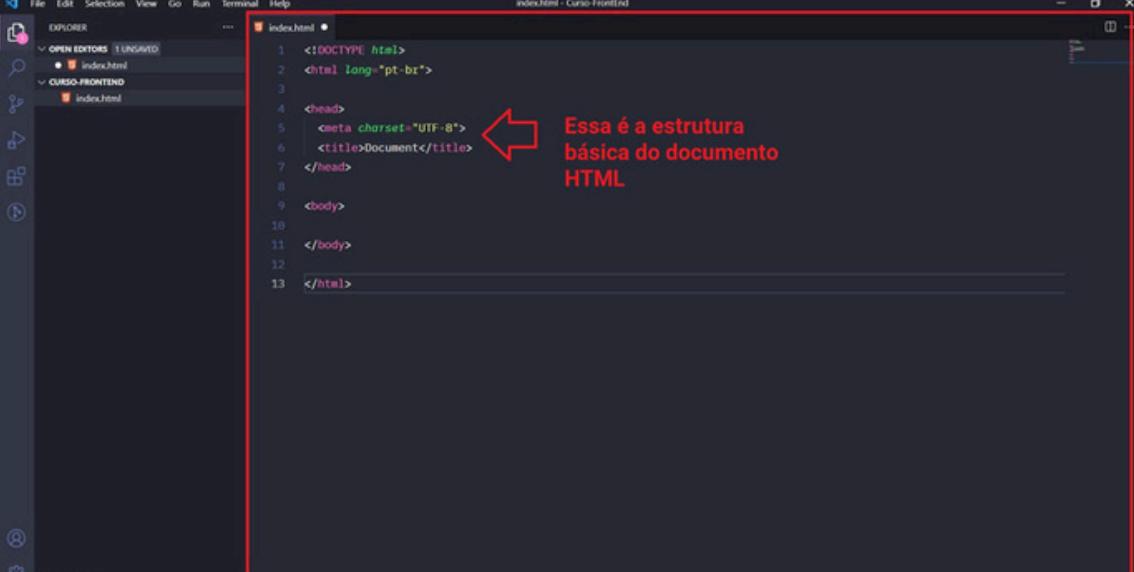
Notem que ao clicarmos sobre o index.html localizado do lado esquerdo da tela (logo abaixo da pasta Curso-FrontEnd) automaticamente do lado direito da tela, surgiu o mesmo arquivo index.html. O lado direito é o local onde vamos digitar todo o código que faz parte da estrutura desse arquivo index.html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



Vamos criar a nossa estrutura básica do Documento HTML que ficará conforme imagem da próxima página.



Colocando a Extensão Live Server no Ar(Pra funcionar)

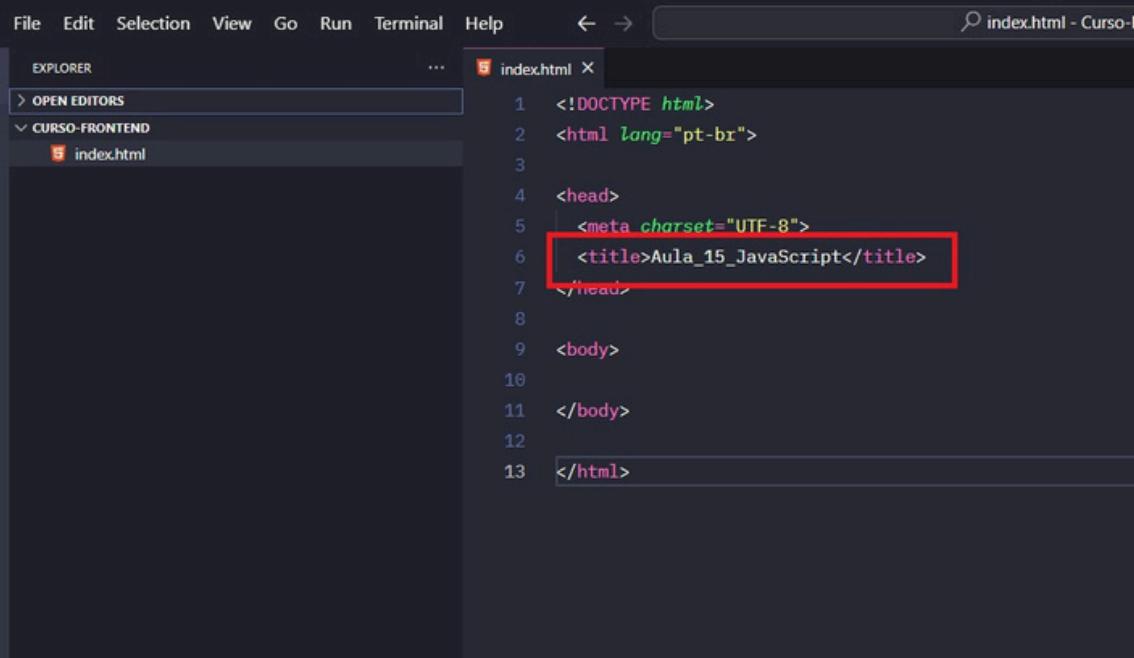


A screenshot of the Visual Studio Code interface. On the left is the Explorer sidebar with a tree view showing 'OPEN EDITORS' (1 unsaved), 'CURSO-FRONTEND' (index.html), and a search bar. The main area is titled 'index.html - Curso-Frontend'. It contains the following code:

```
<!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 tag of the title element.

Vamos dar o nome de Aula_15_JavaScript dentro do <title></title>.. ficando assim: <title>Aula_15_JavaScript</title>



A screenshot of the Visual Studio Code interface, similar to the previous one but with a modification. The title element now contains the text 'Aula_15_JavaScript'. The code is:

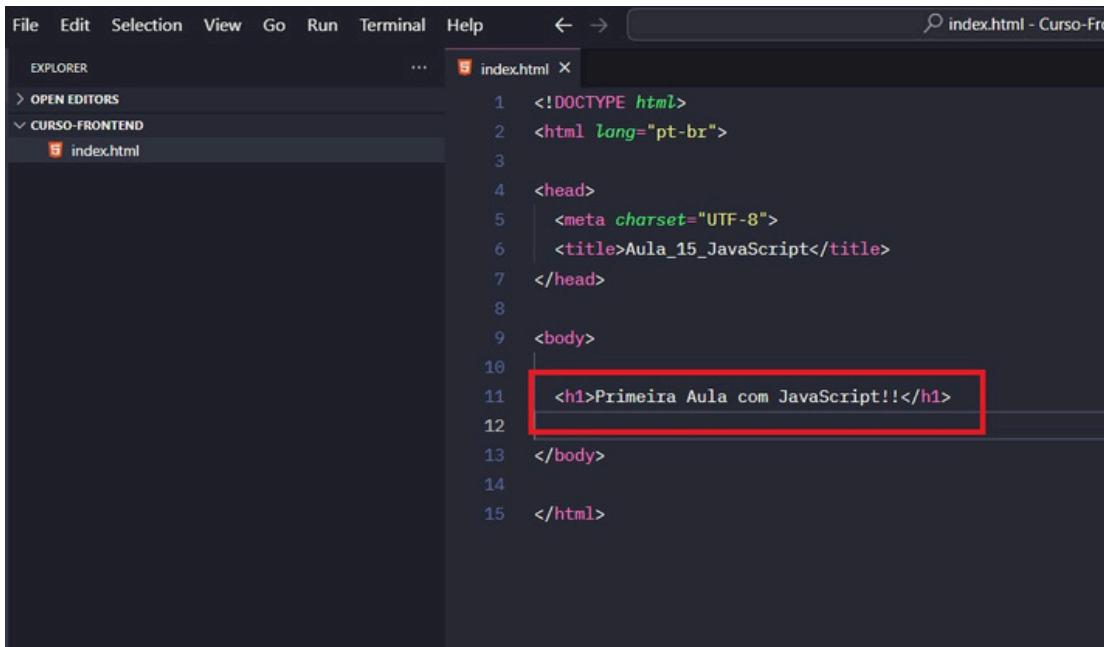
```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Aula_15_JavaScript</title>
</head>
<body>
</body>
</html>
```



/igor-rebolla

Colocando a Extensão Live Server no Ar(Pra funcionar)

Vamos criar um h1 com o nome de Primeira Aula com JavaScript!!
E Bora salvar essas modificações com o bom e velho CTRL + S



A screenshot of the Visual Studio Code interface. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The top right shows a search bar with 'index.html - Curso-Fro'. The Explorer sidebar shows 'OPEN EDITORS' and 'CURSO-FRONTEND' with 'index.html' selected. The main editor area displays the following HTML code:

```
File Edit Selection View Go Run Terminal Help ← → index.html - Curso-Fr  
EXPLORER  
> OPEN EDITORS  
CURSO-FRONTEND  
index.html  
  
index.html x  
1 <!DOCTYPE html>  
2 <html lang="pt-br">  
3  
4 <head>  
5   <meta charset="UTF-8">  
6   <title>Aula_15_JavaScript</title>  
7 </head>  
8  
9 <body>  
10  
11   <h1>Primeira Aula com JavaScript!!</h1>  
12  
13 </body>  
14  
15 </html>
```

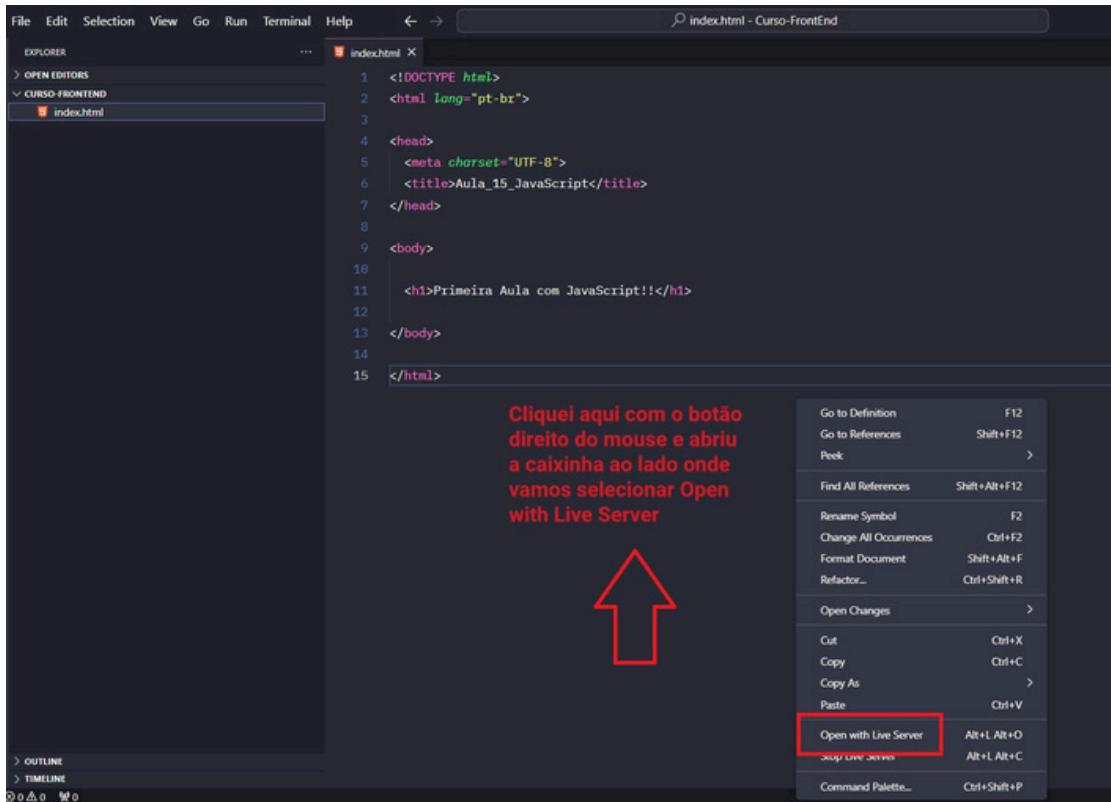
The line containing the new h1 tag ('11 <h1>Primeira Aula com JavaScript!!</h1>') is highlighted with a red rectangular box.

Com o arquivo Salvo, vamos ver como ficou esse arquivo index.html dentro do Navegador(Chrome)

Para ver como ficou precisamos levantar um Servidor Local(Local Server) e esse servidor será criado com a Extensão Live Server, se clicarmos com o botão direito em qualquer lugar que não tenha texto dentro do documento index.html que acabamos de alterar, podemos ver dentre as opções apresentadas um carinha chamado Open With Live Server



Colocando a Extensão Live Server no Ar(Pra funcionar)



Vamos clicar com o botão esquerdo do mouse nessa opção Open With Live Server e ao fazermos isso... de forma automática o arquivo index.html será visualizado dentro do Navegador(Chrome)

Ao abrirmos o Navegador, o que alteramos mais acima dentro do <h1> será exibido, no caso aqui: Primeira Aula com JavaScript!!



Colocando a Extensão Live Server no Ar(Pra funcionar)



O interessante do Live Server é que se efetuamos qualquer modificação no código e salvarmos ele é exibido no Navegador de forma instantânea.

Agora vamos prestar bastante atenção, pois é importantíssimo... o endereço que aparece no campo de pesquisa do navegador.

Tá prestando atenção... Tá mesmo... Olha lá, einh!!

Nós temos um número birutão e na sequência um arquivo index.html, esse arquivo index.html foi aquele que criamos um pouquinho mais acima.

Um pouquinho mais atrás, foi comentado que o endereço que iríamos utilizar tem o nome de LOCALHOST que mostra que estamos utilizando um Servidor Local(Local Server) no nosso computador. Esse número e o localhost são a mesma coisa. Se alterarmos o número 127.0.0.1 para localhost, podemos ver que a mesma página é exibida no Navegador(Chrome).



Colocando a Extensão Live Server no Ar(Pra funcionar)

Com o número birutão + o arquivo index.html



Com o localhost + o arquivo index.html



A mesma mensagem é exibida no navegador.



Sugestões de prática:

1. Refazer e Entender os exemplos dessa aula;
2. Mudar o conteúdo dentro do h1 para um texto de sua preferência, salvar essa alteração e com o live server rodando ver no navegador se o texto é alterado de forma instantânea.

Programação é prática, curiosidade e repetição!!!!



Como adicionar o JavaScript em uma Página Web

Anteriormente vimos como configurar o nosso ambiente com a Instalação do VSCode e do Live Server, além de outro conceito importantíssimo sobre o que é um Servidor.

Dentro do VSCode, vamos usar nesse inicio o mesmo código HTML o qual digitamos anteriormente, ah e não podemos esquecer de inicializar o Live Server.

Agora que o nosso ambiente está preparado, chegou o momento de vermos como adicionamos o JavaScript em uma Página Web.

A primeira coisa que temos que saber é em qual lugar podemos adicionar o JavaScript dentro de uma página HTML.

Podemos utilizar o JavaScript em qualquer local dentro das tags:

- 1 - <head>Fica como Exercício para Pesquisar</head>
- 2 - <body></body>

Nesse primeiro momento não vamos nos preocupar com mais nada, tudo o que vamos fazer é especificar uma tag <script> que contem abertura e fechamento para que possamos escrever o tal do conteúdo JavaScript dentro dessa tag.

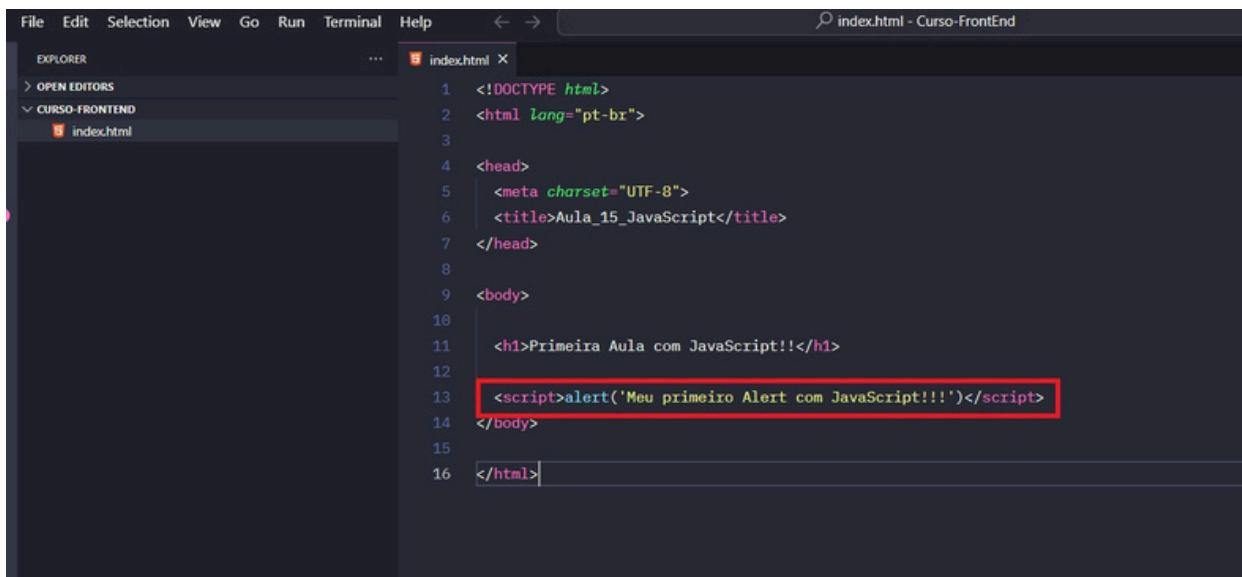


Como adicionar o JavaScript em uma Página Web

Vamos colocar a tag `<script></script>` como a ultima tag dentro da tag `<body></body>`

Então assim podemos, declarar um alert que vai exibir a mensagem ('Meu primeiro Alert com JavaScript!!!').

O código vai ficar desse jeito, conforme imagem abaixo:



```
File Edit Selection View Go Run Terminal Help ← → index.html - Curso-FrontEnd
EXPLORER OPEN EDITORS CURSO-FRONTEND index.html
1 <!DOCTYPE html>
2 <html lang="pt-br">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>Aula_15_JavaScript</title>
7 </head>
8
9 <body>
10
11   <h1>Primeira Aula com JavaScript!!</h1>
12
13   <script>alert('Meu primeiro Alert com JavaScript!!!')</script>
14 </body>
15
16 </html>
```



/igor-rebolla

Como adicionar o JavaScript em uma Página Web

Igor, esse tal de alert com a mensagem dentro dos parênteses, está meio confuso o que seria isso?

Mais acima, eu comentei que não era para nos preocuparmos com mais nada que não fosse entender o local aonde iríamos colocar o JavaScript, pois bem, isso vale aqui também, não se preocupe agora em entender o que esse código significa, mas vou deixar uma breve explicação aqui: ele é apenas um Método que vai gerar um pop-up (uma caixa de diálogo) na tela com uma mensagem e um botão onde podemos clicar em OK (explicaremos isso com mais detalhes nas próximas aulas).

Se salvarmos esse documento que acabamos de alterar com o bom e velho CTRL + S maroto, o código dessa tag JavaScript será executado e vai aparecer uma mensagem com um botão para clicarmos em OK e o restante da página atrás desse pop-up também será carregada.



/igor-rebolla

Como adicionar o JavaScript em uma Página Web

O Exemplo visto na página anterior o qual colocamos a tag <script> no <body> é conhecido como código JavaScript inline. Esse meio é válido quando temos pouquíssimo código, já se tivermos uma quantidade maior de código JavaScript o ideal é que seja feito por um outro meio.

Igor, estava pensando aqui, se esse não é o meio recomendado quando temos uma quantidade maior de código JavaScript dentro da tag <body> então podemos criar um arquivo separado em JavaScript assim como fizemos com o CSS quando estudamos sobre CSS?

Beleza, tá legal, o Igor vai embora, vocês não precisam mais desse professor aqui.. vou vender minha arte na praia.

Ai que orgulhoooooooooooo... é isso mesmo pessoal, vocês estão certíssimos... Podemos isolar esse código em um arquivo separado e só fazermos uma referência desse arquivo JavaScript o qual fora criado separado lá dentro do arquivo index.html, chamamos isso de (Linkar).

Já falei que eu estou orgulhoso, né? Fica mais uma vez o registro!!!

Então bora criar esse arquivo JavaScript separado...

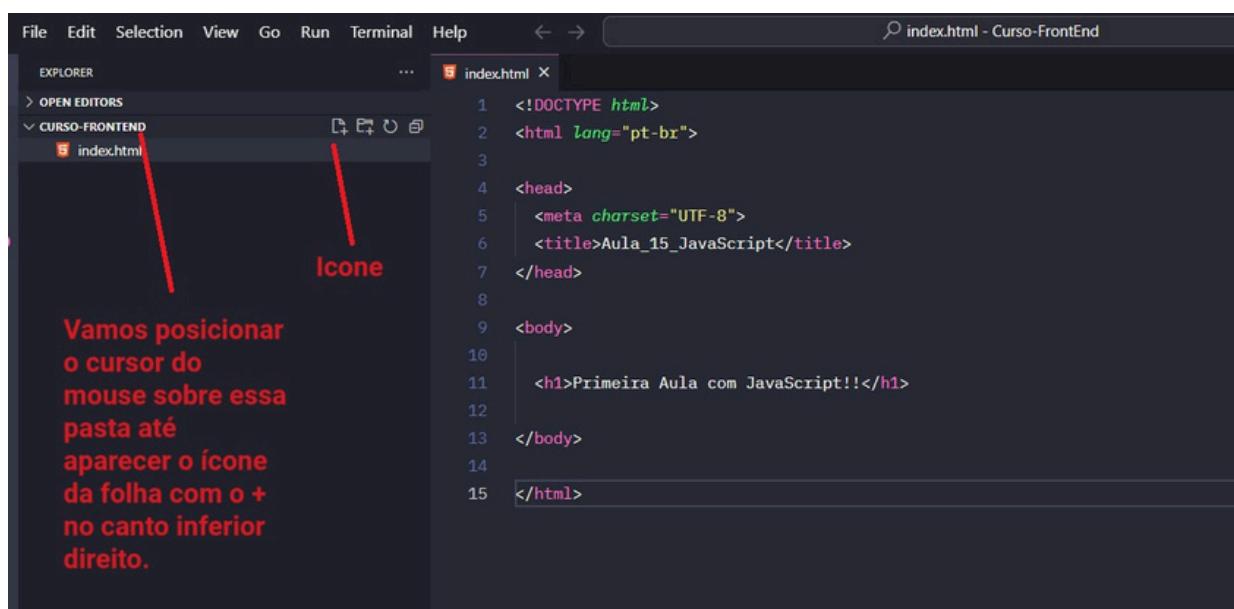


Como adicionar o JavaScript em uma Página Web

A nomenclatura do arquivo é composta por: nome.extensão onde nome(vamos chama-lo de: app) e a extensão depois do . temos que usar a extensão javascript(representada por: js). Então com base nessa definição o nome do nosso arquivo javascript vai ficar assim:

app.js

Agora que definimos o nome do nosso arquivo JavaScript, vamos na lateral esquerda dentro do VSCode, posicionaremos o cursor do mouse sobre a pasta: CURSO-FRONTEND até aparecer alguns ícones a direita.



/igor-rebolla

Como adicionar o JavaScript em uma Página Web

E clicaremos no primeiro item (o desenho de uma folha com um sinal de + no canto inferior direito com o botão direito sobre ela e selecionaremos criar novo arquivo

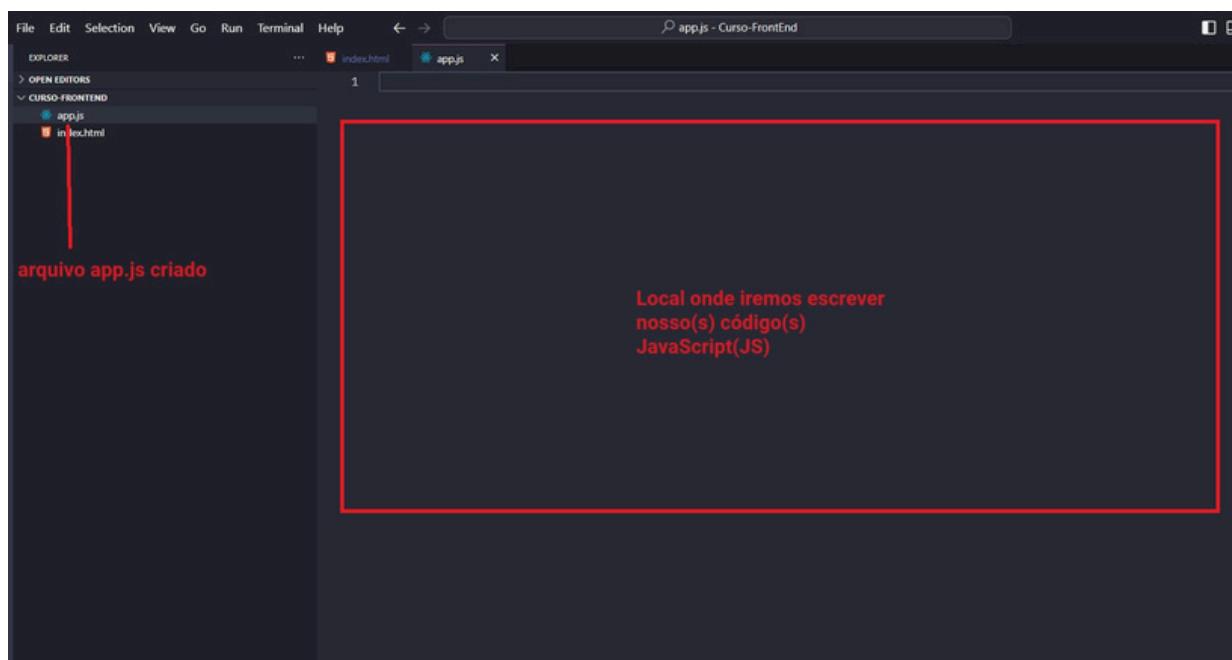
```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Aula_15_JavaScript</title>
</head>
<body>
<h1>Primeira Aula com JavaScript!!</h1>
</body>
</html>
```

Digitaremos app.js e pressionaremos enter.



Como adicionar o JavaScript em uma Página Web

Notem que assim como criamos o arquivo index.html, o arquivo app.js consta abaixo da nossa pasta Curso-FrontEnd e do lado direito abriu um local sem nada escrito, onde vamos escrever todo o nosso código JavaScript.

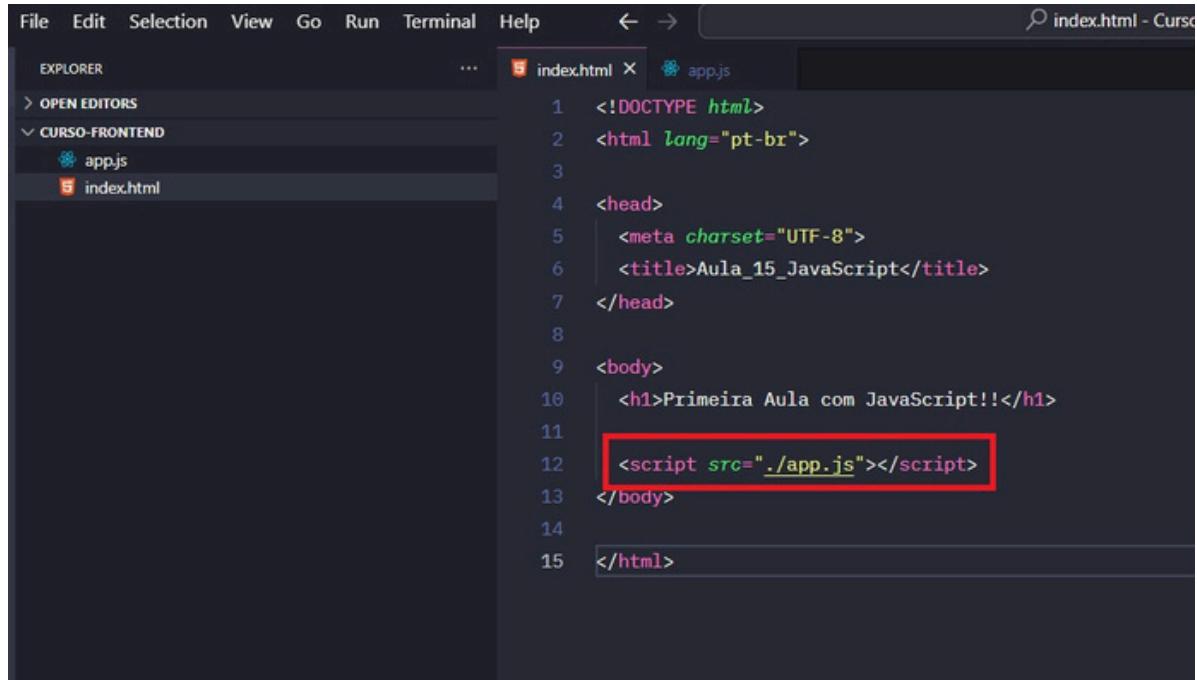


Lembra que para que o arquivo app.js(Agora temos o nome) funcione dentro do index.html temos que fazer um link(Linkar), pois bem vamos fazer isso agora

Abriremos o arquivo index.html, e antes do fechamento da tag </body>, nós vamos declarar uma nova Tag <script></script>, só que aqui ela não vai ter conteúdo igual quando definimos o alert com a mensagem no exemplo mais acima, e sim declararemos um atributo src que vai receber o caminho do arquivo o qual queremos fazer referência(linkar) que é o app.js. Ficando conforme imagem da próxima página.



Como adicionar o JavaScript em uma Página Web



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

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Aula_15_JavaScript</title>
</head>
<body>
<h1>Primeira Aula com JavaScript!!</h1>
<script src="./app.js"></script>
</body>
</html>
```

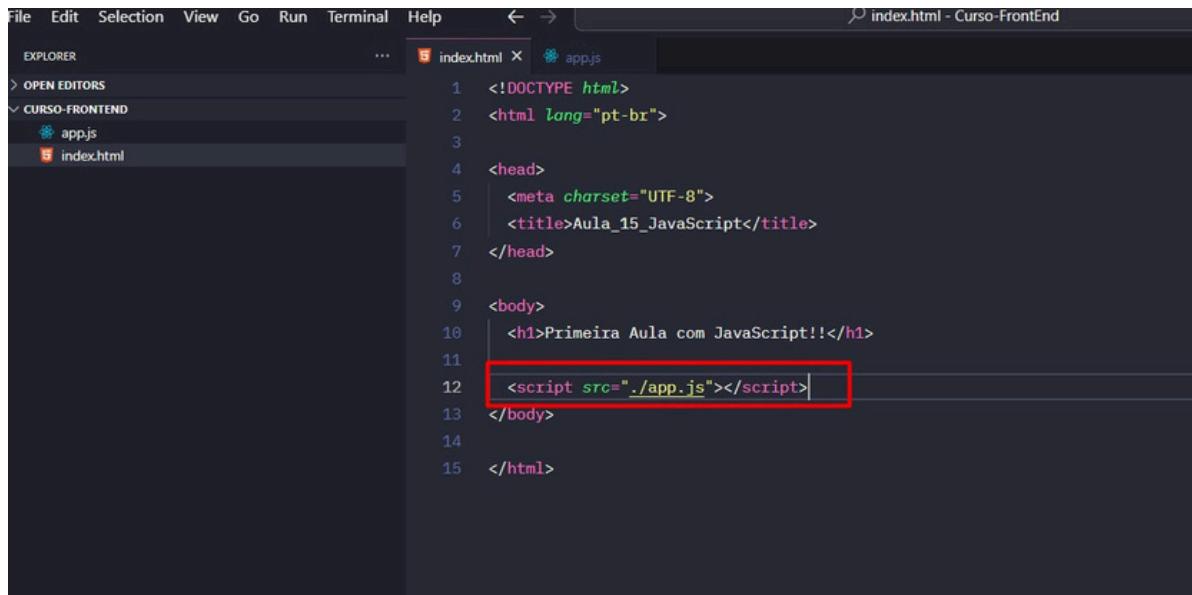
The line containing the script tag is highlighted with a red rectangle.

Com o app.js devidamente referenciado(Linkado) dentro do arquivo index.html, se agora formos escrever código javascript dentro do arquivo app.js, esse código será executado quando rodarmos o arquivo index.html no navegador(Chrome).

Digitaremos: alert('Meu primeiro Alert com JavaScript!!!') e salvaremos esse arquivo mais uma vez com o CTRL + S maroto.



Como adicionar o JavaScript em uma Página Web



A screenshot of a code editor (VS Code) showing the file `index.html`. The code contains an `<script src="../app.js"></script>` tag highlighted with a red rectangle. The code editor interface includes a top bar with File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The left sidebar shows an Explorer with files `app.js` and `index.html`.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Aula_15_JavaScript</title>
</head>
<body>
<h1>Primeira Aula com JavaScript!!</h1>
<script src="../app.js"></script>
</body>
</html>
```



É desse jeito que adicionamos o JavaScript(JS) dentro de uma Página WEB.



O Console do Navegador

O Console é como uma linha de comando inteligente e rica dentro do DevTools e é uma ótima ferramenta complementar para usar com outras ferramentas.

O Console fornece uma maneira poderosa de criar scripts, inspecionar a página da Web atual e manipular a página da Web atual usando JavaScript.

Fonte: <https://learn.microsoft.com/pt-br/microsoft-edge/devtools-guide-chromium/console/>

Nós estamos usando o Google Chrome como navegador padrão aqui no curso, a definição acima foi retirada da página do Microsoft Edge, e isso foi proposital, para vocês verem que essa definição vale tanto para o: Chrome, Edge e FireFox.

Para abrirmos essa ferramenta temos 2 jeitos para fazermos:

- 1 - Pressionar F12 no teclado com o Navegador(Chrome) aberto
- 2 - Clicar com o botão direito em qualquer lugar que não tenha nada escrito dentro do Navegador(Chrome) e clicar com o botão esquerdo do mouse em Iinspecionar e selecionaremos a aba Console

Mostraremos a segunda forma para abrirmos a Ferramenta



O Console do Navegador

Igor, não entendi nada do conceito técnico sobre o Console do Navegador (DevTools, Inspecionar, Manipular e etc...) poderia explicar de uma outra forma?

É pra já e é ai que vem a parte Lúdica da coisa:

Podemos considerar a aba Console dentro do DevTools(Ferramentas do Desenvolvedor) como um parque de diversões do Desenvolvedor o qual temos disponível no Navegador(Chrome) para testar, inspecionar e manipular o código JavaScript.

O DevTools nós usaremos tanto mais tanto e mais tanto(eu já disse mais tanto né) que ele vai ser quase como um membro da familia, por um pequeno detalhe que nós não poderemos declará-lo no imposto de renda como dependente.

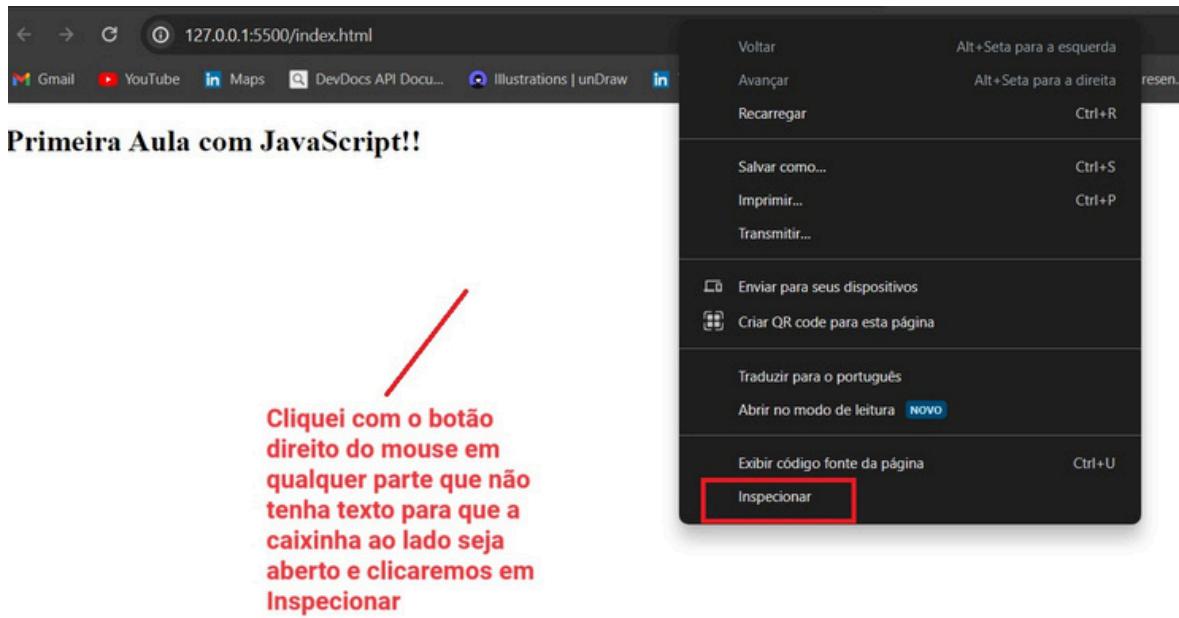
DevTools esse é o Pessoal, Pessoal esse é o DevTools... Agora que vocês já foram devidamente apresentados, vamos ver como esse carinha funciona na prática.

Ah, ops... já ia me esquecendo... antes precisamos abrir esse carinha....

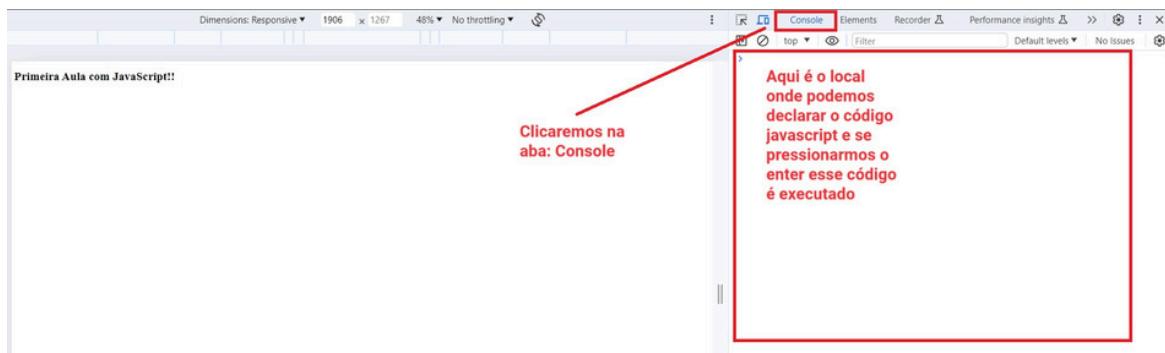
Clicar com o botão direito em qualquer lugar que não tenha nada escrito dentro do Navegador(Chrome) e clicar com o botão esquerdo do mouse em Inspecionar e selecionaremos a aba Console



O Console do Navegador



Dentro da Aba Console, podemos declarar código JavaScript e ao pressionarmos o Enter esse código JavaScript será executado.

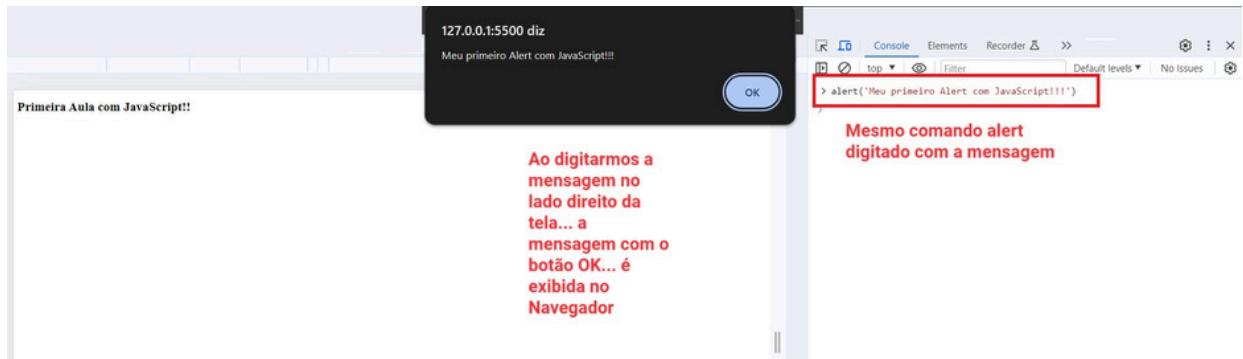


Ao declararmos o mesmo comando o qual usamos no exemplo um pouquinho mais acima: alert('Meu primeiro Alert com JavaScript!!!') e pressionarmos enter a mensagem com o botão OK, será mostrada no Navegador(Chrome).



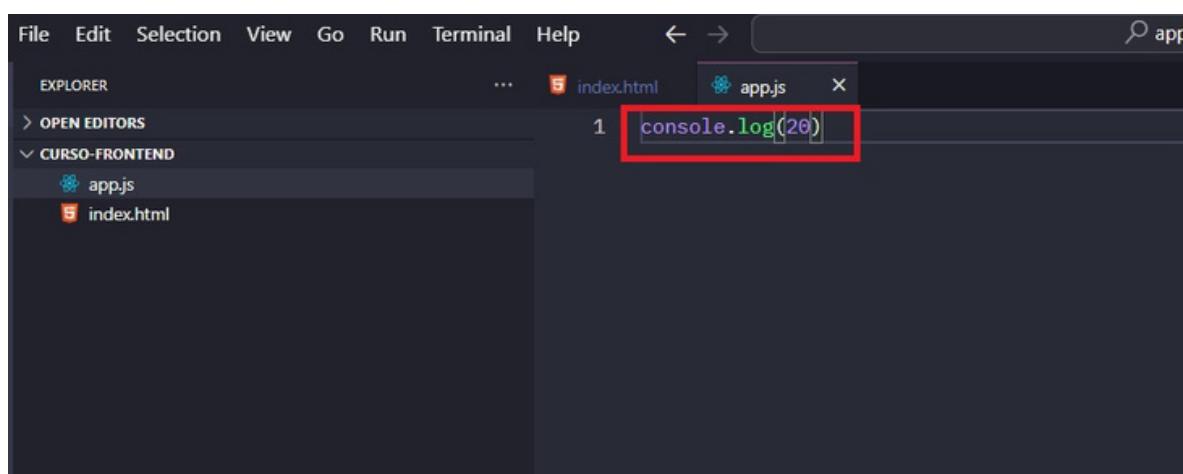
/igor-rebolla

O Console do Navegador



Além de fazermos isso, também é possível que seja mostrado algo desse Console que vem laaaaaaaaaaaa do nosso arquivo app.js

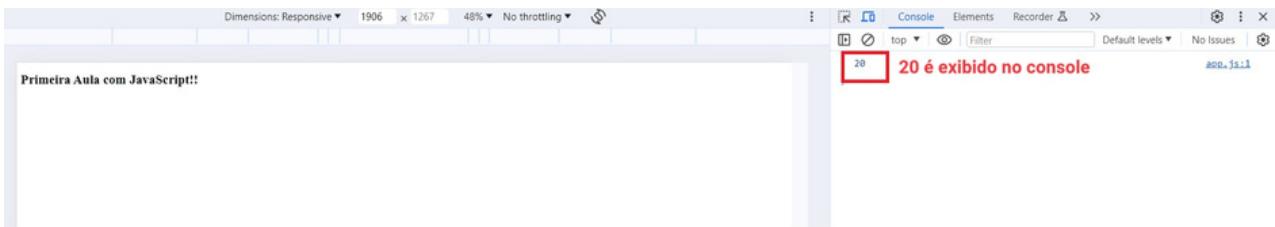
Removeremos o alert que inserimos no exemplo anterior, vamos digitar console.log() e aqui dentro dos parênteses colocaremos o número 20.



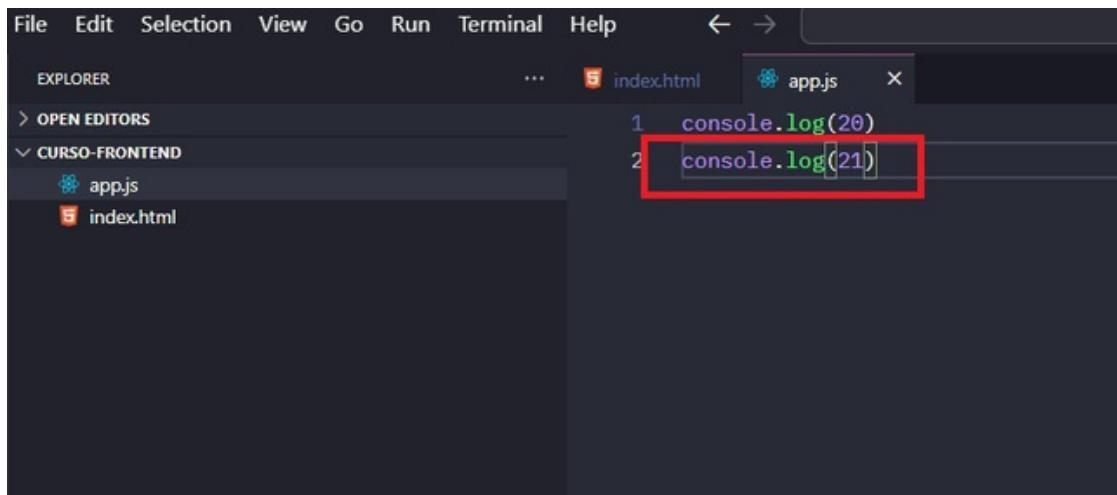
E ao salvarmos com aquele CTRL + S maroto, o número 20 é exibido no console.



O Console do Navegador



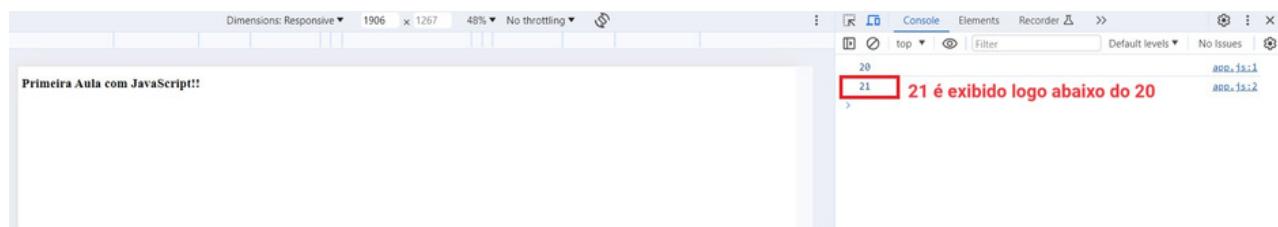
Se logo abaixo do `console.log(20)` digitarmos `console.log(21)`, esse código será executado de modo sequencial, pois o JavaScript é executado (Top-Down) ou seja de cima para baixo.



Então o que acontecerá aqui quando salvarmos esse arquivo e executarmos no navegador(Chrome), a primeira linha será executada e na sequência a segunda linha. Conforme imagem abaixo (onde o console do DevTools vai exibir 20 e logo abaixo 21)



O Console do Navegador



Só reforçando mais uma vez, não vamos ficar preocupados nesse momento com o que significa esse `console.log`, a sua estrutura como `ponto(.)` e `parênteses()`, nas próximas aulas veremos certinho a definição, exemplos e como ele funciona(entre várias outras coisas).

O que eu gostaria que vocês soubessem agora, é que esse `console.log` é um método que podemos usar para exibir valores (caracteres, números...) no console do DevTools e isso vai ser muito útil nesse momento em que vocês estão estudando sobre a linguagem(JavaScript).

Sugestões de prática:

1. Refazer e Entender os exemplos dessa aula;
2. Na página 02 dessa aula eu deixei a tag <head> como pesquisa, peço a gentileza que essa pesquisa seja efetuada, se eu posso colocar a tag script aqui dentro.
3. Alterar os números dentro do console.log(20) e console.log(21) com números da Vossa preferência e verificar o comportamento dessa alteração dentro do DevTools na aba Console.
4. Abrir o DevTools dentro do Navegador(Chrome) pressionando a tecla F12.

Programação é prática, curiosidade e repetição!!!!



O que é essa Tal de Variável ?

Um dos fundamentos mais importantes em uma linguagem de programação vem a ser o conceito de Variáveis.

E o que essa tal de Variável vem a fazer?

O Papel de uma variável é armazenar um valor o qual nós definimos, esse valor pode ser (número, nome, e-mail...).

E como esse valor fica armazenado, podemos sempre usa-lo no local onde julgarmos necessário.

Bora ver como essa variável é declarada



Palavra Chave let

Usaremos a palavra-chave(key word) let

Para ficar mais claro vamos abrir o VS CODE e dentro do arquivo app.js (o qual criamos na aula 16 desse curso)

Vamos declarar uma let com o nome cep que recebe um número 12345
(Estamos falando... “Ilustríssimo deixa essa variável cep armazenar esse número que no caso aqui é o 12345”)

The screenshot shows the VS Code interface with the 'app.js' file open. The code is:

```
1 let cep = 12345
```

Red vertical lines from the bottom point to the words:

- variável
- nome da variável
- valor da variável

Palavra Chave let

Precisamos prestar muita...muita...muita (eu já disse muita né) mas é importantíssimo isso aqui...

O símbolo = que está entre cep e 12345, nós não vamos ler esse símbolo como estamos acostumados no nosso dia a dia que é o igual (=) igual dentro do JavaScript é representado de outra forma o qual veremos mais a frente nesse curso).

Quando olharmos para esse símbolo devemos interpretá-lo como SIMBOLO DE ATRIBUIÇÃO. A forma correta ao lermos essa linha dentro do JavaScript vai ser:

```
let cep = 12345
```

Onde (cep recebe 12345) esse número a partir desse momento está guardado/armazenado dentro da memória e em qualquer outro momento podemos acessá-lo através do nome que especificamos aqui logo depois da variável let que é o cep.

Logo abaixo da linha onde digitamos let cep = 12345, utilizaremos o comando console.log(comando esse que vimos na aula 16 desse curso) usando esse comando, podemos exibir o valor dentro do console do Navegador(Chrome) através do nome da variável.

O comando vai ficar assim:

```
console.log(cep)
```

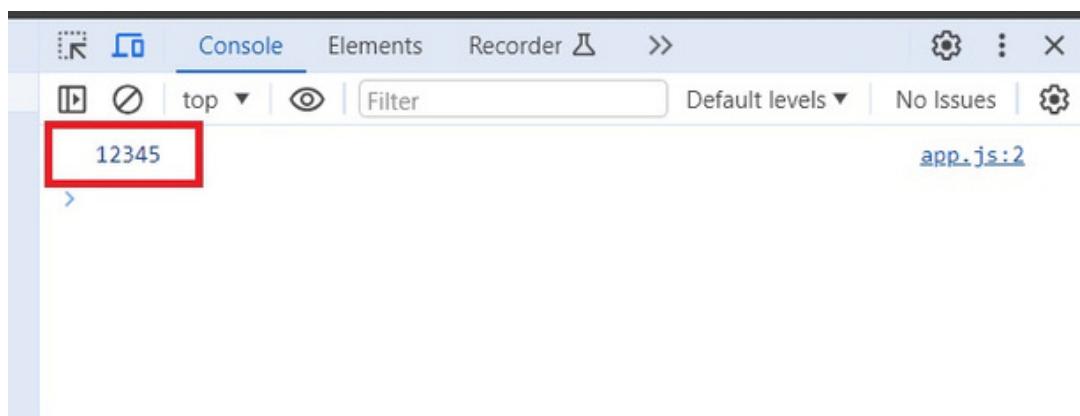


Palavra Chave let

The screenshot shows a dark-themed interface of the Visual Studio Code editor. At the top, a navigation bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help' menus, along with back and forward arrows. Below the menu bar is a toolbar with icons for file operations. The main area is divided into two panes: the left pane, titled 'EXPLORER', lists 'OPEN EDITORS' and a folder 'CURSO-FRONTEND' containing files 'app.js' and 'index.html'; the right pane, titled 'TERMINAL', displays a command-line interface with the following text:

```
1 let cep = 12345
2 console.log(cep)
```

E quando salvamos o arquivo app.js com aquele CTRL + S maroto, o número 12345 é exibido no console.



Palavra Chave Iet

Até o momento vimos que a palavra-chave (KEYWORD) let cria uma variável e SE vem a ser uma variável o valor dela pode variar e isso nos mostra que em outro momento é possível indicarmos um novo valor para essa variável.

E pra vermos como isso funciona, vamos no nosso arquivo app.js e logo abaixo do console.log(cep) o qual digitamos no exemplo acima, vamos escrever só o nome da variável cep sem a palavra let pra indicarmos para o JavaScript(JS) que estamos utilizando essa variável, seguido do sinal de atribuição(=) e por último e não menos importante o novo valor que ela vai receber que é 67890. Ficando assim:

cep = 67890

The screenshot shows the Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. Below the menu is a toolbar with back, forward, and search icons. The left sidebar has sections for EXPLORER, OPEN EDITORS, and CURSO-FRONTEND. Under CURSO-FRONTEND, there are two files: app.js and index.html. The right side is the code editor with the following content:

```
1 let cep = 12345
2 console.log(cep)
3 cep = 67890
```

The line 'cep = 67890' is highlighted with a red box.



/igor-rebolla

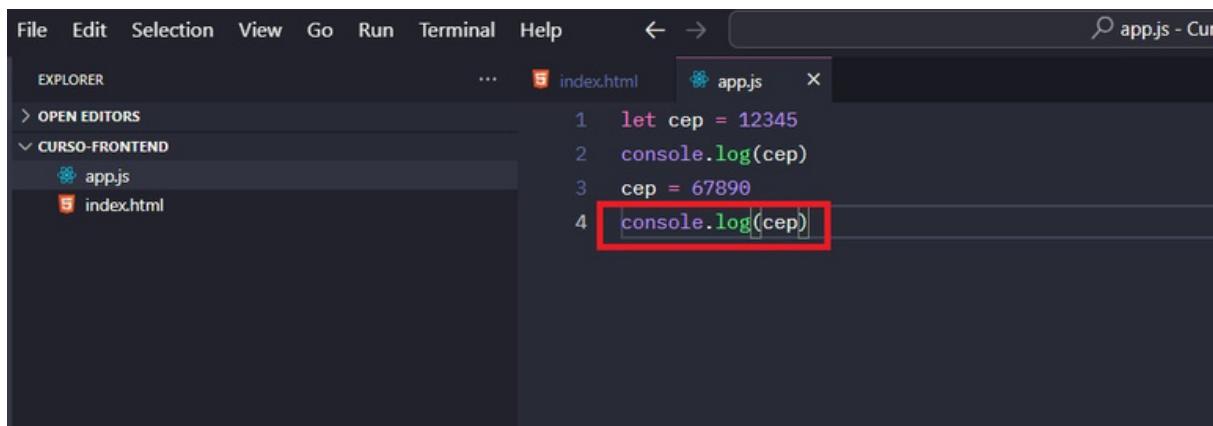
Palavra Chave let

Igor, está meio confuso.. primeiro digitamos o let cep agora só o cep. Pode explicar melhor isso?

Só se for agora... Lembra que na aula 16 eu comentei que o código dentro do JavaScript(JS) é executado de cima para baixo(TOP-DOWN) pois bem, aqui em primeiro lugar o JavaScript(JS) vai ler que essa variável foi declarada lá em cima(TOP) do código e na linha mais abaixo(DOWN) a qual indicamos um novo valor, a partir daqui o valor dessa variável foi atualizado para 67890.

Então percebemos aqui que para efetuarmos uma nova indicação de valor nós não precisamos usar a palavra let novamente, só precisamos escrever o nome da variável a qual já declaramos que no caso aqui é cep e alteramos o valor dela que no caso aqui é 67890

Aqui declararmos um console.log(cep) logo abaixo dessa reatribuição
cep = 67890



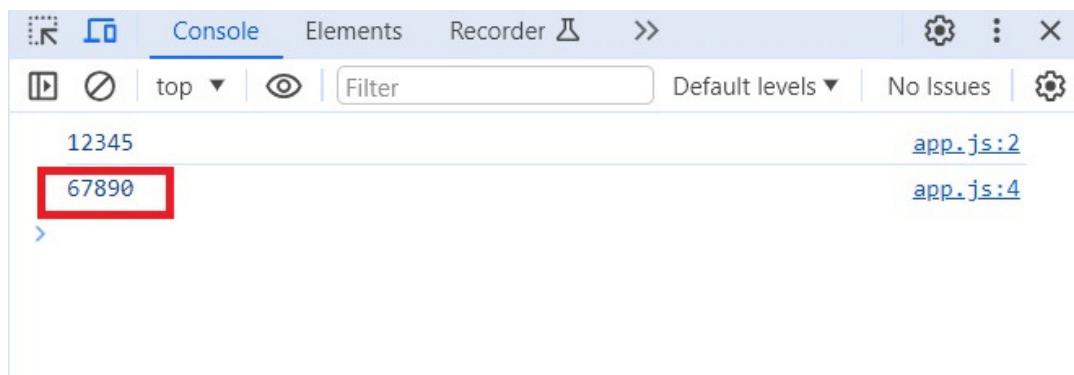
```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Cur...  
EXPLORER ... index.html app.js x  
> OPEN EDITORS  
CURSO-FRONTEND  
app.js  
index.html  
1 let cep = 12345  
2 console.log(cep)  
3 cep = 67890  
4 console.log(cep)
```



/igor-rebolla

Palavra Chave let

E salvarmos esse arquivo com o bom e velho CTRL + S maroto, o primeiro console.log vai exibir a cep com o valor de 12345 que foi atribuído a ela lá em cima(TOP) do código e esse segundo console.log abaixo(DOWN) do novo valor indicado vai exibir 67890



Palavra Chave const

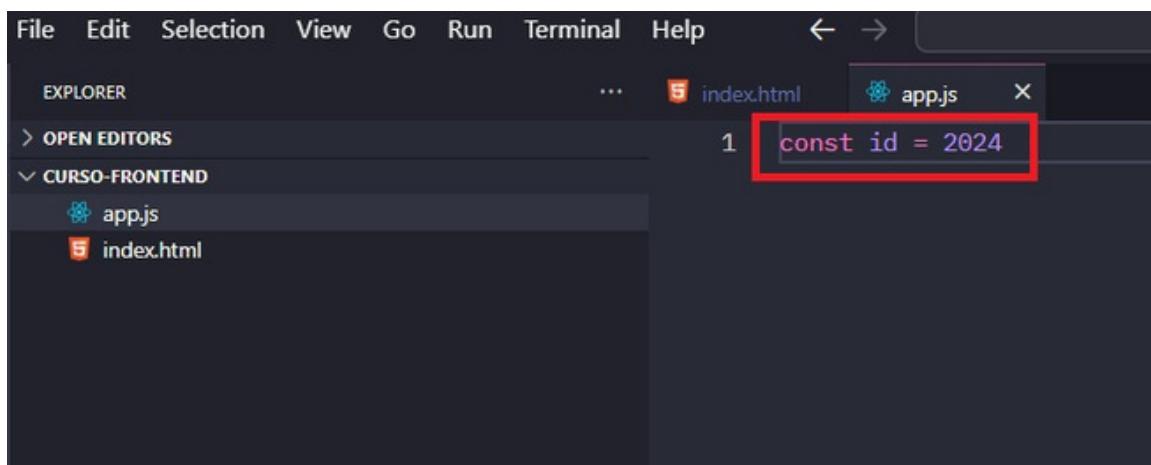
Agora e se quisermos criar uma variável na qual não queremos atribuir um novo valor a ela (Deixa ela lá tranquilona com aquele valor recebido e esse valor fica lá como senão houvesse o amanhã). Para isso vamos usar a segunda palavra-chave(keyword) para criar essa variável que é a:

const.

Assim como fizemos com a palavra-chave(keyword) let, vamos aqui também usar o VS CODE para melhor entendimento.

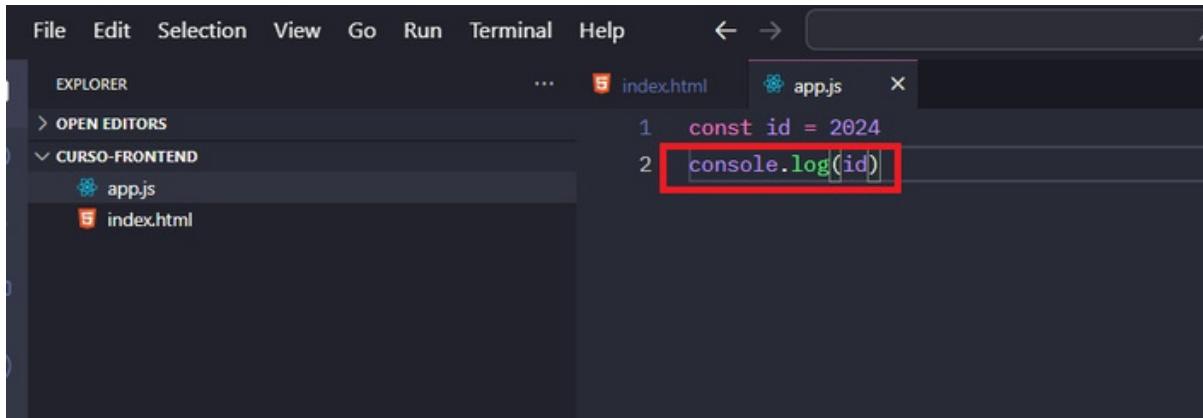
E dentro do nosso arquivo app.js, vamos declarar uma const(fazendo isso será declarada uma constante) nomearemos essa const como id atribuiremos para ela um número 2024. Ficando assim o código:

```
const id = 2024
```



Palavra Chave const

E logo abaixo vamos exibir essa const dentro do Navegador(Chrome) no console e para fazermos isso digitaremos na linha abaixo console.log(id)

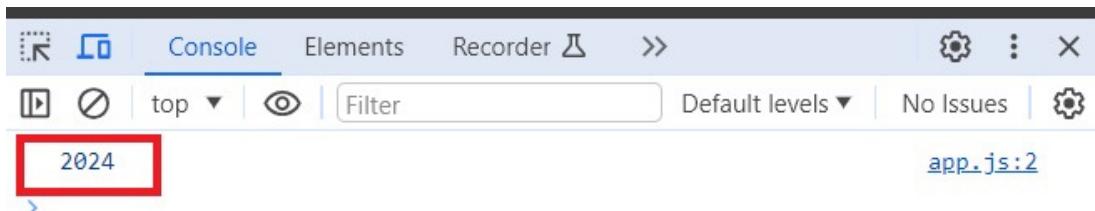


A screenshot of the Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. Below the menu is a toolbar with back, forward, and search icons. The Explorer sidebar shows 'OPEN EDITORS' and a folder 'CURSO-FRONTEND' containing 'app.js' and 'index.html'. The main editor area displays two lines of code:

```
1 const id = 2024
2 console.log(id)
```

The second line, 'console.log(id)', is highlighted with a red box.

E se salvarmos com aquele CTRL + S maroto o número 2024 é exibido no console do Navegador(Chrome)

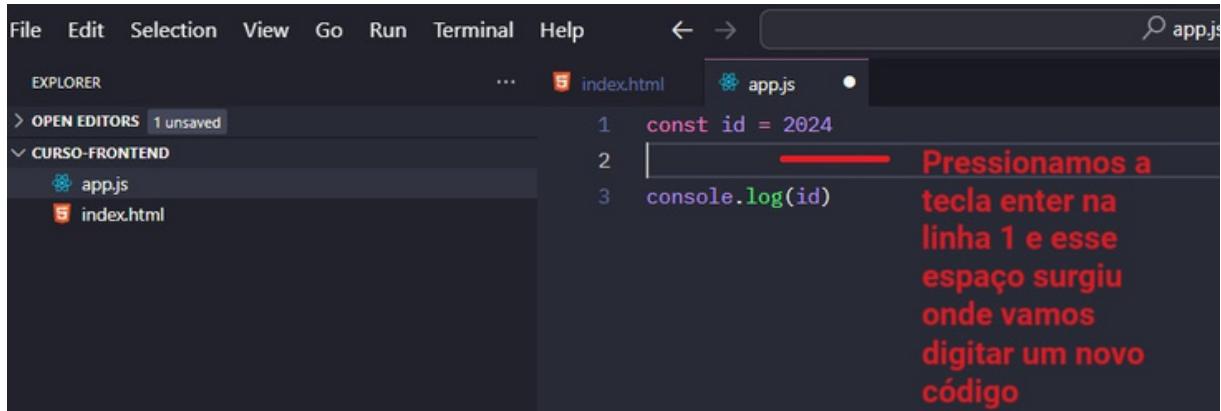


A screenshot of the Chrome DevTools interface, specifically the 'Console' tab. The tab bar also includes 'Elements' and 'Recorder'. Below the tabs are various controls like 'top', 'Filter', and 'Default levels'. The main console area shows the output '2024' in blue text, which is highlighted with a red box. To the right of the output, it says 'app.js:2'.

Igor, e se quisermos alterar o valor dessa const id para 3000, tem como? Vamos ver um exemplo se isso é possível. Bora dar um enter entre a linha(1) const id = 2024 e a linha(2) console.log(id)

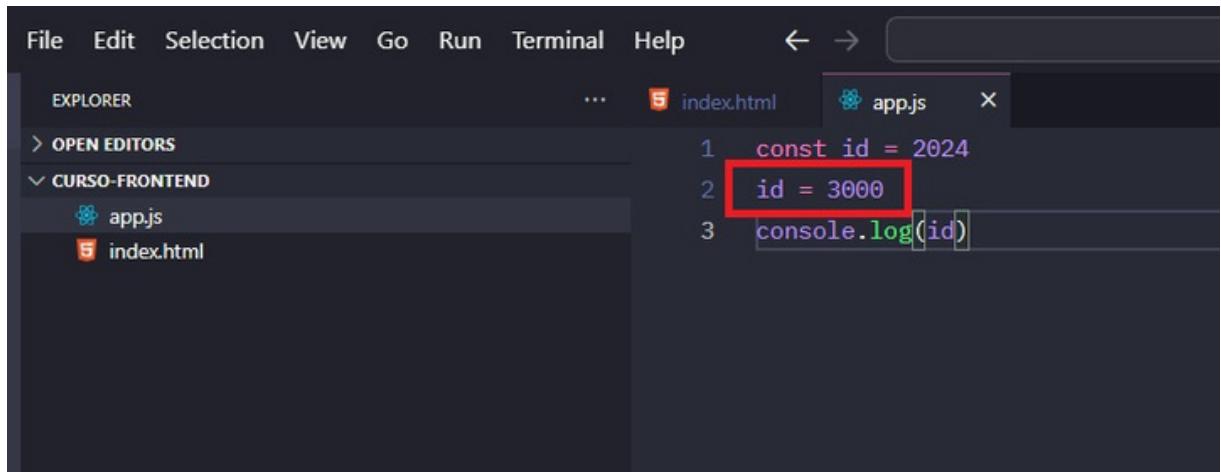


Palavra Chave const



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js  
EXPLORER ... index.html app.js ●  
> OPEN EDITORS 1 unsaved  
CURSO-FRONTEND  
app.js  
index.html  
1 const id = 2024  
2 | _____ Pressionamos a  
3 console.log(id) tecla enter na  
linha 1 e esse  
espaço surgiu  
onde vamos  
digitar um novo  
código
```

E entre essas 2 linhas vamos digitar id e reatribuir um novo número informado por vocês na pergunta com a dúvida que no caso aqui é 3000. Ficando assim: id = 3000



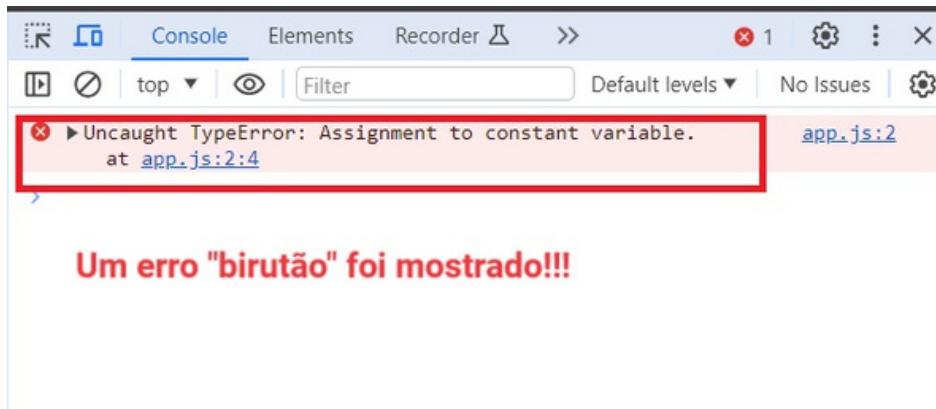
```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js X  
EXPLORER ... index.html app.js  
> OPEN EDITORS  
CURSO-FRONTEND  
app.js  
index.html  
1 const id = 2024  
2 id = 3000  
3 console.log(id)
```

Vamos salvar o arquivo com aquele CTRL + S maroto

E que rufem os tambores.... será que o valor 3000 será exibido no console do Navegador(Chrome)..... bora ver então:



Palavra Chave const



Eita, foi mostrado um erro "Birutão" no console, dizendo que nós tentamos atribuir um novo valor para uma constante(`const`).

Lembra que a `const` só quer ficar de boa na lagoa com aquela primeiro valor que ela recebeu, pois bem é esse erro que dá quando ousamos incomoda-la com um novo valor.

Bora dar aquela reforçada, vamos usar `const` quando não quisermos que outro valor seja atribuído para essa `const` ou seja quando quisermos que esse valor da declaração da `const` seja constante em toda a aplicação. E se esse não for o cenário, nós usamos `let` para indicarmos novos valores

As palavras-chaves (`let` e `const`) são formas mais atuais que foram adicionadas no JavaScript(JS) para a criação de variáveis.



Palavra Chave var

Existe uma forma em desuso para criarmos uma variável que é utilizando a palavra-chave(keyword):

var



Do mesmo jeito como fizemos com let e const, vamos aqui também usar o VSCODE para melhor entendimento.

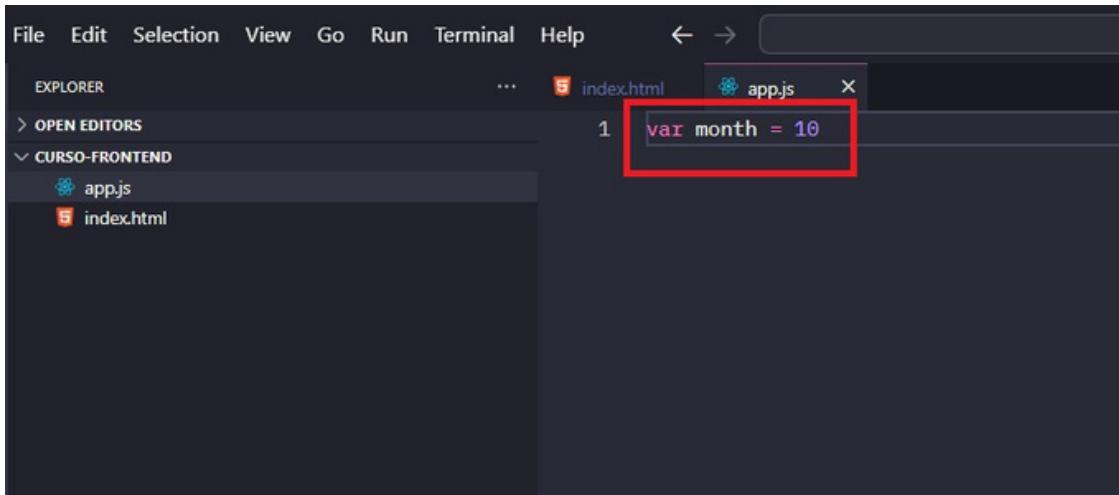
Assim como os Maias e os Astecas (Zoeira hehehehe)...

Antigamente (era essa palavra que eu queria lembrar) se quiséssemos criar uma variável month com o valor 10, nós precisávamos declarar essa variável com a palavra-chave(keyword) var com o nome month que receberia um número no caso aqui 10



Palavra Chave var

E dentro do nosso arquivo app.js, vamos declarar essa var com o exemplo acima. Ficando assim:

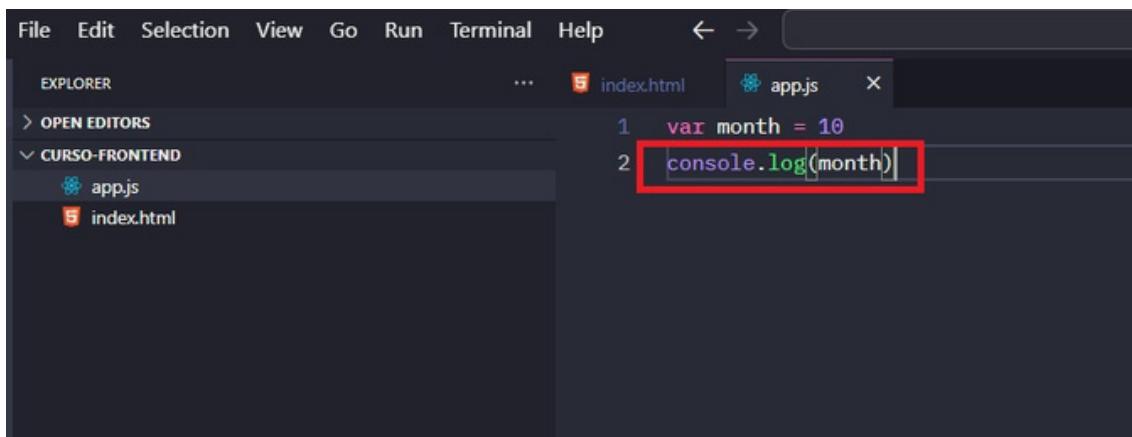


A screenshot of the Visual Studio Code interface. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The Explorer sidebar shows an open folder 'CURSO-FRONTEND' containing files 'app.js' and 'index.html'. The main editor tab is 'app.js', which contains the following code:

```
1 var month = 10
```

The line 'var month = 10' is highlighted with a red rectangle.

E para exibirmos no console esse valor, usaremos o comando `console.log(month)`



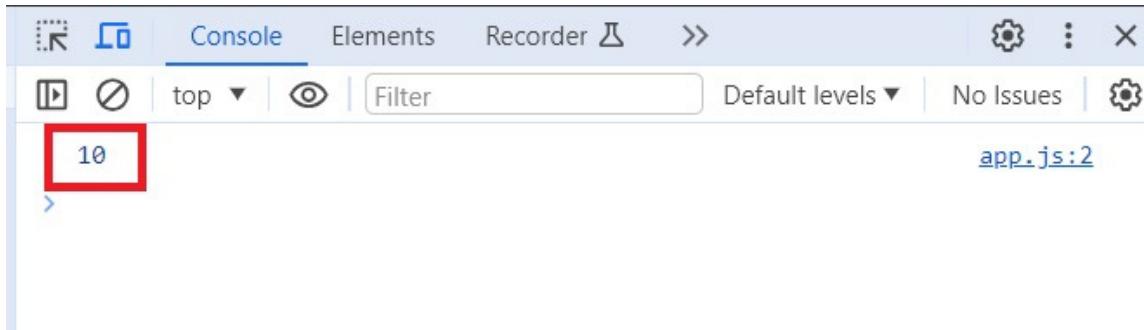
A screenshot of the Visual Studio Code interface. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The Explorer sidebar shows an open folder 'CURSO-FRONTEND' containing files 'app.js' and 'index.html'. The main editor tab is 'app.js', which contains the following code:

```
1 var month = 10
2 console.log(month)
```

The line 'console.log(month)' is highlighted with a red rectangle.

Palavra Chave var

E ao salvarmos esse arquivo com o CTRL + S maroto o valor 10 é exibido no console



Eu mostrei a var para vocês, caso algum dia você se depare com um código mais antigo, você terá o conhecimento base do que a var faz, que também armazena um valor e assim como a let ela aceita a indicação de um valor.

Existem mais detalhes sobre o var que pode ser pesquisado na internet, mas como aqui é um curso no qual eu quero mostrar o que tem de mais atual na linguagem, vamos utilizar let e const.

Uma última frase sobre o var:

var não deveria ser usado nem no futebol!!!

Desligando o modo quinta série B em: 3..2...1

3 regrinhas para nomearmos variáveis

1º Nomes de variáveis só podem conter (Números, Letras, Cifrão(\$) e Underscore(_)), porem o nome não deve iniciar com um número

2º Existem alguns nomes de variáveis que não podem ser utilizados, porque eles já são uma palavra-chave(keyword) reservada do JavaScript(JS)

Exemplo: não é possível declararmos uma let que se chama let
(let let = 200)

3º Uma variável deve ser uma palavra exclusiva. Isso nos mostra que não é possível que uma variável venha a se chamar: street number

The screenshot shows a dark-themed code editor with a red box highlighting a syntax error in the file 'app.js'. The error is at line 1, character 13, where there is a space between 'let' and 'street'. A red line points from this error to a explanatory text on the right. Another red line points from the explanatory text to the space between 'street' and 'number'.

Nem salvamos o arquivo e ele já indicou um erro, o erro indicado é justamente o espaço entre as palavras

Esse espaço entre as palavras não é permitido

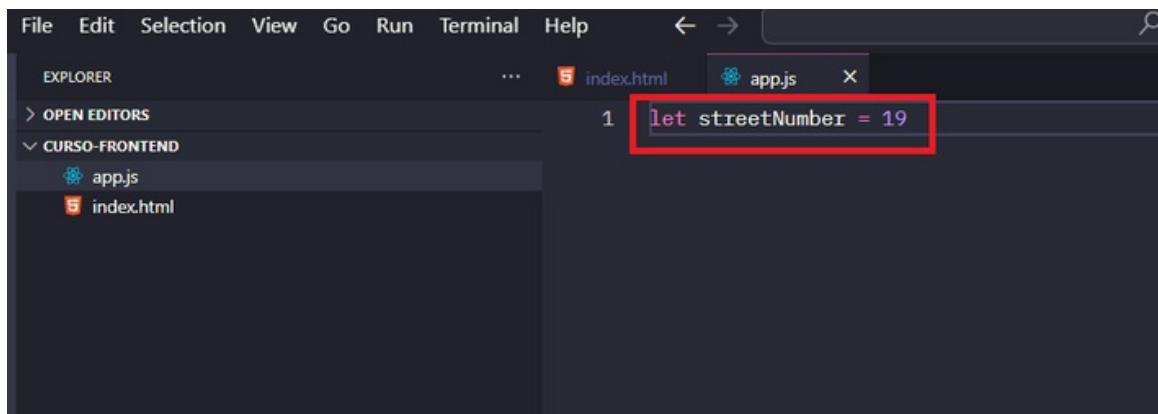
```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND app.js index.html
app.js
index.html
app.js - Curso-FrontEnd
1 let street number = 19
```



3 regrinhas para nomearmos variáveis

Se quisermos usar duas ou mais palavras para nomearmos uma variável existe uma convenção com o nome de CAMELCASE, como essa convenção funciona.

Se eu quiser declarar street number por exemplo. A primeira coisa que eu tenho que fazer é juntar essas 2 palavras, ficando streetnumber, nós vamos manter a primeira letra(s) da primeira palavra(street) em minúscula e a primeira letra(n) da segunda palavra(number) vou transforma-la em maiúscula. Ficando assim let streetNumber = 19



```
File Edit Selection View Go Run Terminal Help ⌘ ⌘ ↶ ↷ 🔎
EXPLORER ... index.html app.js ✎
> OPEN EDITORS
CURSO-FRONTEND
  app.js
  index.html
1 let streetNumber = 19
```



/igor-rebolla

3 regrinhas para nomearmos variáveis

Aqui como estamos aprendendo o JavaScript(JS) e ainda no inicio desse aprendizado podemos "brincar" com os nomes das variáveis, agora conforme vamos evoluindo o recomendado é usar nomes que façam algum sentido nas variáveis (pois assim outras pessoas que vão ler o seu código consiga entender de forma clara) chamamos isso de boas práticas dentro da programação.

Já pensou você tá lá fazendo um teste para uma vaga de emprego e define uma variável para let boladaoDoSupino = 200 (Não é uma boa prática nem se você tiver fazendo o teste para entrar na Smart Fit por exemplo) Por isso bons nomes são recomendados.



Sugestões de prática:

1. Refazer e Entender os exemplos dessa aula;
2. Criar o nome de uma variável com 4 palavras seguindo a convenção CamelCase igual exemplo do streetNumber e exibir o valor no console do Navegador(Chrome)
3. Seguindo as regrinhas para nomearmos uma variável, é possível criar uma variável let chamada const e atribuir um valor a ela?
Colocar isso em prática no VSCode e ver o resultado.

Programação é prática, curiosidade e repetição!!!!



Uma visão geral sobre tipos de dados

Anteriormente nós vimos como os valores são armazenados em uma variável, só os valores que nós vimos foram apenas formados por Numbers(Números), E os Numbers(Números) que nós utilizamos nos exemplos estavam relacionados com: cep, id, month e streetNumber.

E o tipo Number(Número) é só o primeiro dos tipos de dados em JavaScript(JS). Em JavaScript nós podemos trabalhar com até 8 tipos de dados. Listaremos cada um deles separadamente, mostrando suas particularidades.

- E só reforçando o primeiro tipo de dado é o que nós já vimos que é o Number(Número)... 9,10,11, 8.90 4567 ah e os números decimais também fazem parte do tipo Number(Número)
- O segundo tipo de dados são as boas e velhas Strings. Strings nada mais são que uma cadeia de caracteres carinhosamente chamados de textos mas para que possamos representar textos dentro do JavaScript(JS), temos que representa-los dentro de aspas duplas ou simples...

Aspas duplas ("Strings") - Exemplo: "aula18@aula18.com"

Aspas simples ('Strings') - Exemplo: 'Salve, galera'

- O terceiro tipo de dado é o nobre Boolean. Booleans são lógicas que representam valores verdadeiro(s) ou falso(s) (true / false) e usaremos esse tipo de dado para validar se determinadas condições são verdadeiras ou falsas (true / false).



/igor-rebolla

Uma visão geral sobre tipos de dados

- O quarto tipo de dado é o Null. O Null é um meio para falarmos explicitamente que uma variável não vale nada ou seja não tem valor, podemos criar uma variável e atribuir um Null a ela, para apontar que essa variável ainda não tem nenhum valor
- O quinto tipo de dado é o Undefined. O Undefined parece com o Null só que é diferente (parece brincadeira, mas é isso mesmo). A diferença é que o Undefined é colocado automaticamente pelo JavaScript(JS)

OBS IMPORTANTE: Tanto o Null como o Undefined representam valores vazios, só que o NULL é utilizado pela pessoa que tá fazendo o desenvolvimento e o Undefined é colocado de forma automática pelo JavaScript(JS).

OBS IMPORTANTE 2 - A missão: Peço a gentileza para quem estiver estudando por esse material, para não se preocupar com Undefined, Null, Condição entre outros termos nesse momento, pois conforme vamos evoluindo com as aulas, esses tipos de dados farão cada vez mais sentido para vocês. Confia no tio!!!

- O sexto tipo de dado é o OBJECT. OBJECT são estruturas de dados um pouco mais robustas e dentro dessas estruturas podemos ter (funções e várias propriedades).

Existem diversos tipos de Objetos diferentes dentro da linguagem JavaScript(JS) os quais podemos utilizar, eles são: (Arrays, Objetos Literais ,Funções e Datas).



/igor-rebolla

Uma visão geral sobre tipos de dados

E não menos importante temos os 2 últimos tipos de dados (BIGINT e o SYMBOL), o qual falaremos brevemente aqui abaixo para que vocês tenham conhecimento sobre os mesmos.

- O sétimo tipo de dado é o BIGINT, o BigInt é utilizado para simbolizar números inteiros “grandão”.
- O oitavo tipo de dado é o SYMBOL, o Symbol é uma adição mais recente dentro do JavaScript(JS) que está relacionado aos objetos

Então nas próximas aulas estudaremos (Number, String, Boolean, Null, Undefined e Arrays com mais detalhes).

Uma variável nós já vimos o que é, uma variável pode armazenar diferentes tipos de dados (Uma string, um número e etc...). Eu não gostaria que de forma alguma você se sinta carregado com muita informação ou venha a achar que terá que guardar todos esses tipos de dados, não foca nisso, por favor.

Isso aqui é só um resumo bem rápido para que você possa saber que existem tipos de dados na linguagem e isso é tudo o que você necessita conhecer até esse momento . E com o passar das aulas e do tempo quanto mais você conseguir praticar no seu dia a dia, essas informações surgirão de forma automática pra ti.

Por isso eu sempre falo:

Programação é prática, curiosidade e repetição!!!!



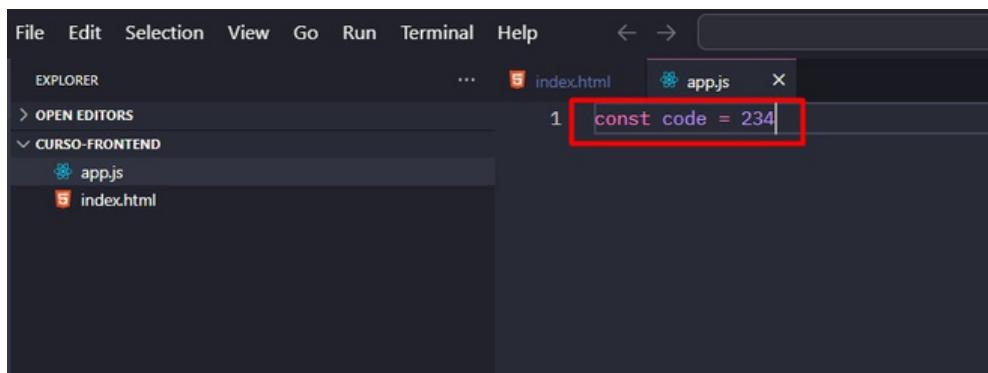
Comentários

Vamos abrir o nosso VSCode para melhor entendimento, colocar o Live Server no Ar e usaremos o nosso arquivo app.js

Digamos que você queira fazer algum tipo de comentário nesse código para que outra pessoa faça a leitura desse código, você pode fazer isso, criando um comentário de uma única linha: usando duas barras e em seguida o seu comentário, ficando assim: // Esse é o meu primeiro comentário de uma única linha

Digitaremos uma const code que recebe 234, ficando assim:

```
const code = 234
```

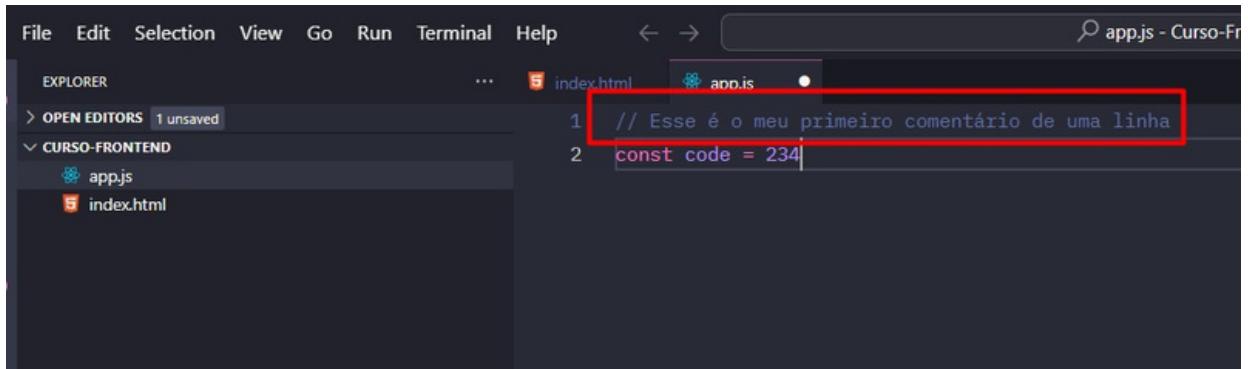


Vamos colocar o cursor do mouse entre o número 1 e o letra "c' da palavra const, pressionaremos o enter a assim vai criar um espaço onde digitaremos o nosso comentário, ficando assim:

```
// Esse é o meu primeiro comentário de uma linha  
const code = 234
```



Comentários



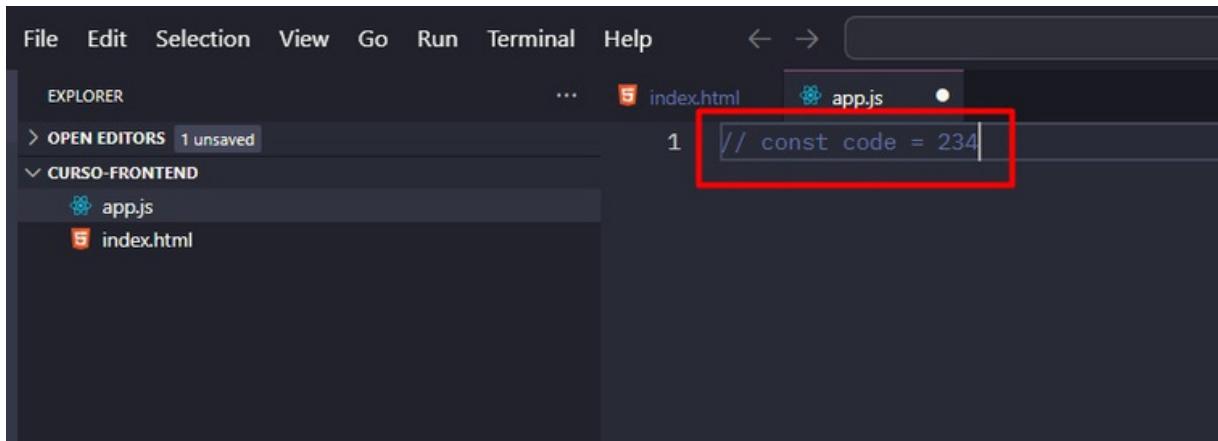
A screenshot of the Visual Studio Code interface. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The status bar shows 'app.js - Curso-Frontend'. The Explorer sidebar shows 'OPEN EDITORS 1 unsaved' and a folder 'CURSO-FRONTEND' containing 'app.js' and 'index.html'. The main editor tab is 'app.js' with the following code:

```
1 // Esse é o meu primeiro comentário de uma linha
2 const code = 234
```

The first line is highlighted with a red rectangle.

É possível também comentarmos o código inserindo `//` antes do código, para isso apagaremos o comentário de uma linha do exemplo acima `// Esse é o meu primeiro comentário de uma linha` e colocaremos: `//` antes do `const code = 234`, ficando assim:

```
// const code = 234
```



A screenshot of the Visual Studio Code interface, identical to the previous one but with a different code snippet in the editor. The main editor tab is 'app.js' with the following code:

```
1 // const code = 234
```

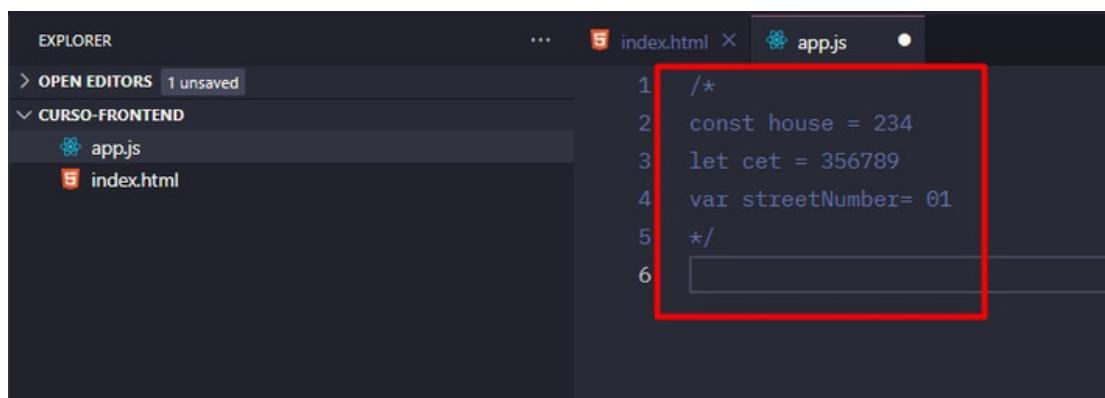
The entire line is highlighted with a red rectangle.

/igor-rebolla

Comentários

Conseguimos também escrever comentários de várias linhas, onde podemos quebrar uma linha e escrevermos um bloco de texto dentro dele, para isso, basta abrirmos o comentário com uma barra e um asterisco /* Colocar o comentário aqui!!! E fechar o comentário com um asterisco barra */ como fizemos quando aprendemos o CSS. Ficando assim:

```
/*
const house = 234
let cet = 356789
var streetNumber= 01
*/
```



The screenshot shows a code editor interface with a dark theme. On the left, the Explorer sidebar lists 'OPEN EDITORS' (1 unsaved) and a folder 'CURSO-FRONTEND' containing 'app.js' and 'index.html'. The main editor area displays the following code:

```
1  /*
2  const house = 234
3  let cet = 356789
4  var streetNumber= 01
5  */
6
```

A red rectangular box highlights the multi-line comment block from line 1 to line 5.

Também é possível comentarmos **blocos de código** com esse tipo de comentário e assim esse código não será executado pelo JavaScript(JS)

Igor, mas essa explicação sobre comentários foi muito rápida, é isso mesmo?

Vou responder essa pergunta com um comentário de uma linha até para ir praticando

// Sim



Primeiro tipo de dado: Numbers

Na aula 17 quando aprendemos sobre variáveis, vimos um pouco sobre Numbers e agora falaremos mais detalhadamente referente aos Numbers

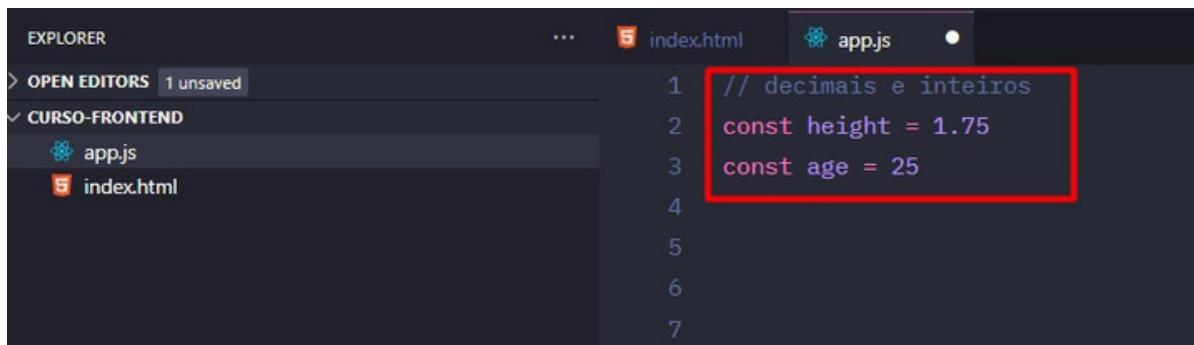
Existem inúmeras situações que os Numbers podem ser usados no JavaScript e agora veremos alguns desses casos

Como eu já estou com o meu VsCode aberto, o Live Server marotinho no ar e o arquivo app.js no jeito, bora ver esses Numbers Danadinhas de forma mais detalhada.

Abaixo do comentário: // decimais e inteiros criaremos 2 consts(constantes), a primeira const se chamará height(altura) que recebe 1.75 e a segunda se chamará age(idade) que recebe 25, ficando assim:

```
const height = 1.75  
const age = 25
```

IMPORTANTE SABER: Para representarmos números decimais em JavaScript(JS) nos utilizamos o ponto (.) ao invés de vírgula (,)



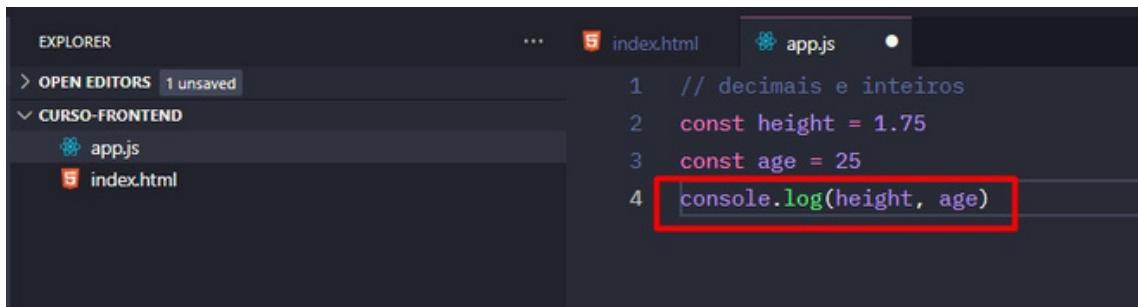
```
// decimais e inteiros  
const height = 1.75  
const age = 25
```



/igor-rebolla

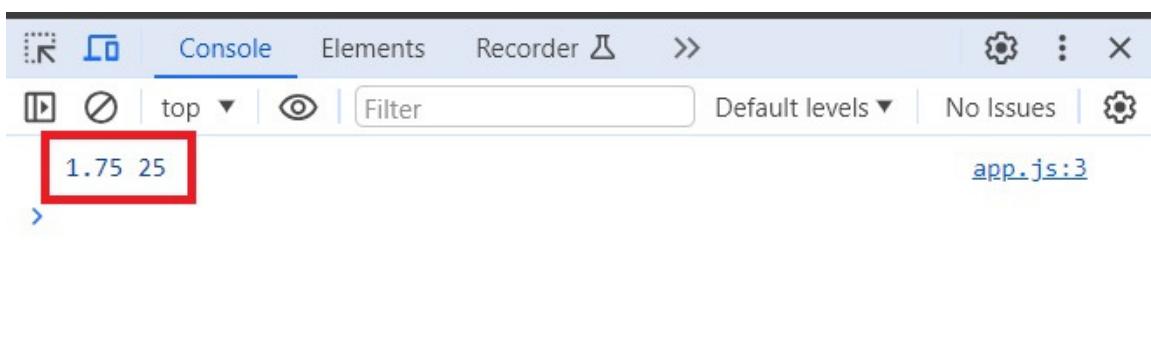
Primeiro tipo de dado: Numbers

E agora abaixo dessas 2 consts declararemos um console.log(height, age).



```
// decimais e inteiros
const height = 1.75
const age = 25
console.log(height, age)
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que os 2 valores são exibidos lado a lado. Onde o primeiro valor é um número decimal e o segundo valor é um número inteiro.



```
1.75 25
```

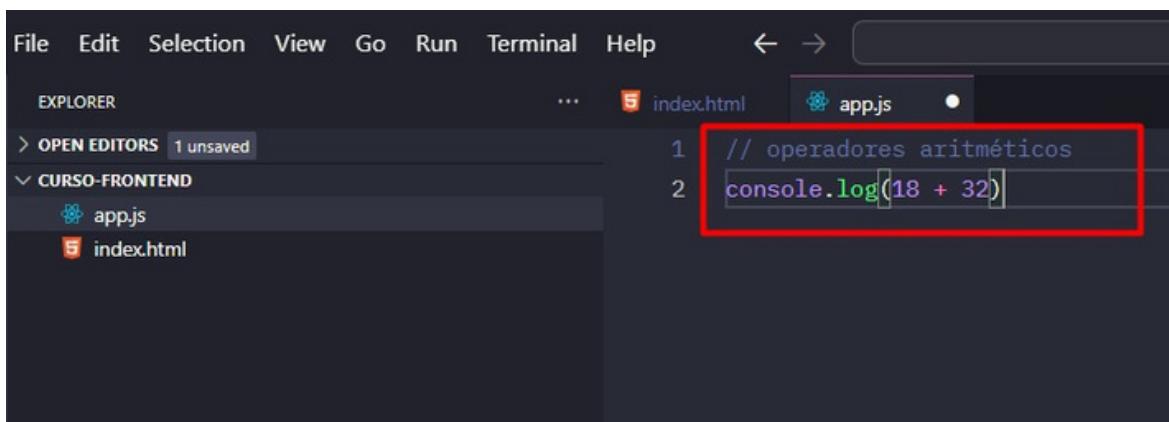


Primeiro tipo de dado: Numbers

Agora que vimos tanto os números decimais quanto os números inteiros, veremos o que é possível fazer “magicamente” com os números no JavaScript(JS). De forma resumida: todas as operações matemáticas, tais como: (adição, subtração, divisão, multiplicação, potencia e etc...)

Abaixo do comentário // operadores aritméticos vamos inserir um console.log e passar 18 + 32, ficando assim:

```
console.log(18 + 32)
```

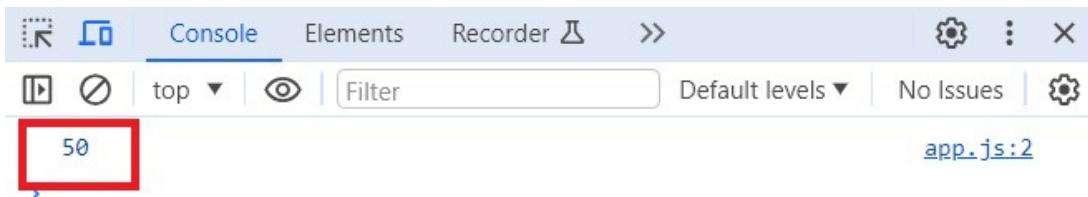


```
File Edit Selection View Go Run Terminal Help ← →
EXPLORER ...
> OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // operadores aritméticos
2 console.log(18 + 32)
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o número 50 será exibido.

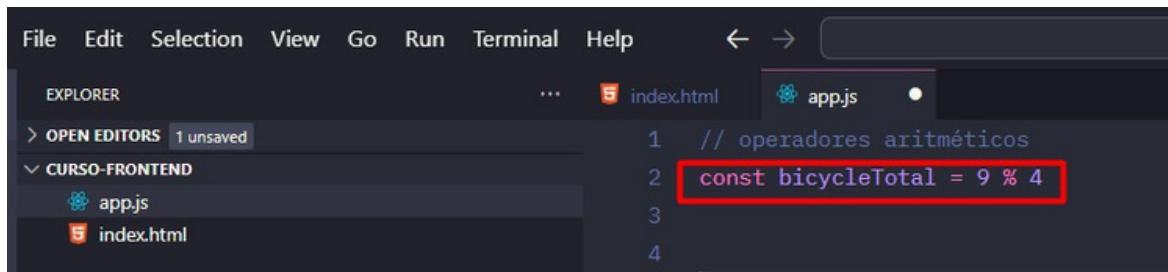


Primeiro tipo de dado: Numbers



E ainda Abaixo do comentário // operadores aritméticos vamos declarar uma const bicycleTotal que recebe 9 módulo 4, ficando assim:

```
const bicycleTotal = 9 % 4
```

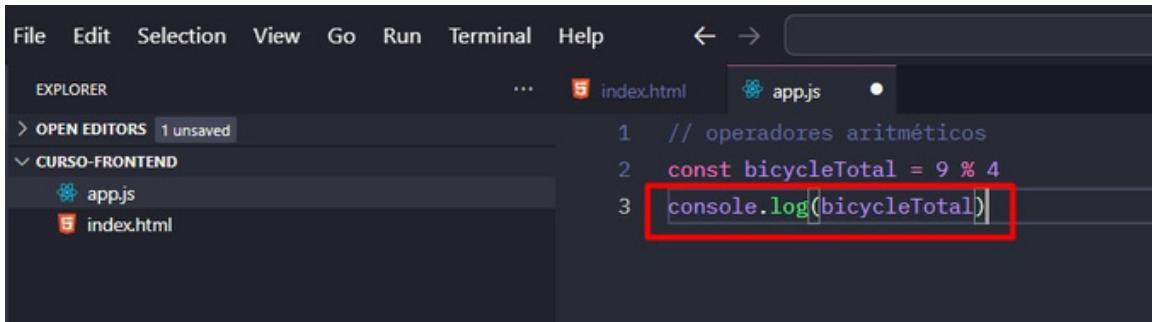


Digitaremos um console.log passando a bicycleTotal, ficando assim:

```
console.log(bicycleTotal)
```



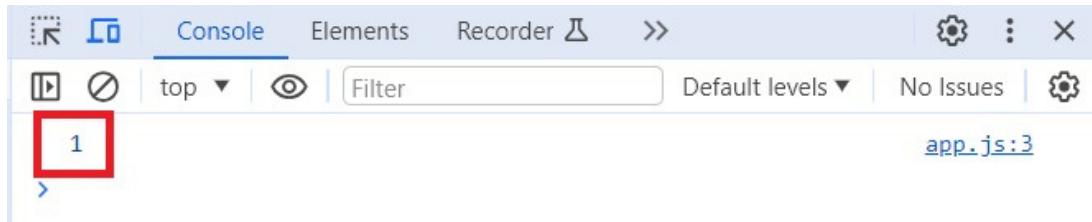
Primeiro tipo de dado: Numbers



```
File Edit Selection View Go Run Terminal Help ← →
EXPLORER ... index.html app.js
> OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // operadores aritméticos
2 const bicycleTotal = 9 % 4
3 console.log(bicycleTotal)
```

The screenshot shows a code editor interface with a dark theme. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The Explorer sidebar shows an open editor for 'CURSO-FRONTEND' containing 'app.js' and 'index.html'. The status bar indicates '1 unsaved'. The main code area contains three lines of JavaScript: a comment, a variable assignment, and a call to 'console.log'. The third line, '3 console.log(bicycleTotal)', is highlighted with a red rectangle.

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto poderemos ver no console do Navegador(Chrome) que o valor 1 é exibido.



Igor, ficou confuso muito confuso mesmo... Módulo, Símbolo de Porcentagem que recebe determinado valor, poderia explicar melhor? Claro, o operador modulo aqui dentro do JavaScript(JS) é representado pelo símbolo de Porcentagem... ele recebe o resto da divisão entre 2 números.

Vou dar um exemplo, para ficar ainda mais claro.... bora ver na próxima página.



Primeiro tipo de dado: Numbers

O nome da variável bicycleTotal foi pensado cuidadosamente para melhor entendimento....Se você tem 9 bicicletas e divide essas bicicletas entre 4 amiguinhos. Com quantas bicicletas cada amiguinho fica?

Você responderia e eu confesso que também responderia: Ah.... com 2 bicicletas e meia, só que não podemos cortar uma bicicleta ao meio ou seja nesse exemplo vai sobrar uma bicicleta inteira, então o resto dessa operação será 1.

E logo abaixo do comentário // ordem das operações matemáticas vamos falar um pouco sobre ordem de precedência entre operações que é a mesma forma que a ordem da precedência básica da matemática, resumindo: quando uma expressão tem mais de uma operação você pode obter respostas diferentes dependendo da ordem que a expressão é resolvida, por isso que existe uma ordem de precedência determinada para execução das operações que é a seguinte:

1º ordem de precedência determinada: Parênteses

2º ordem de precedência determinada: Raízes ou Potências

3º ordem de precedência determinada: Multiplicação e Divisão

4º ordem de precedência determinada: Adição e Subtração



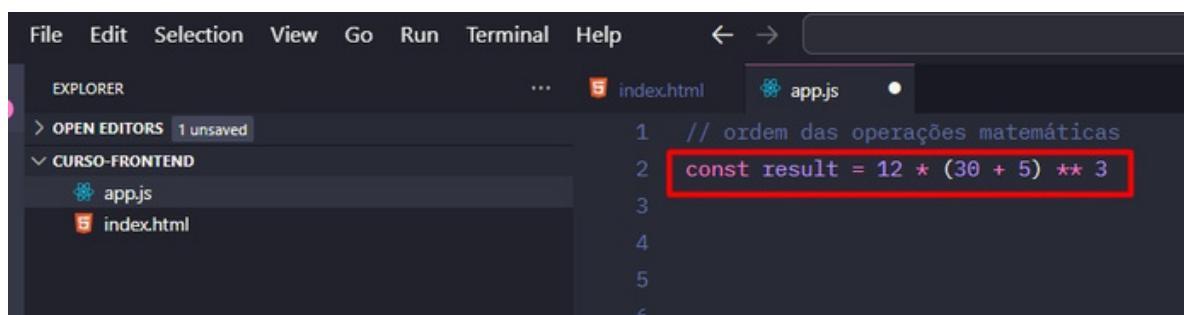
Primeiro tipo de dado: Numbers

Resumindo: Antes de qualquer coisa o JavaScript(JS) verificará se uma expressão tem parênteses, caso não tenha parênteses ela vai procurar uma raiz ou uma potência, caso também não tenha nenhuma raiz e nenhuma potência ela procurará por multiplicação e divisão, caso também não tenha nenhuma multiplicação e divisão... Calma que tá acabando... UFA.... e por último e não menos importante a boa e velha adição e subtração caso ela encontre uma adição ou subtração ela vai resolver isso.

Ainda abaixo do comentário // ordem das operações matemáticas, declararemos uma const result que recebe doze vezes, abre e fecha parênteses e dentro desses parênteses vamos inserir $30 + 5$ e fora desses parênteses vamos eleva-lo ao cubo, ficando assim:

```
// ordem das operações matemáticas  
const result = 12 * (30 + 5) ** 3
```

Obs.: A operação digitada acima mostra que essa ordem clássica de precedência prevalecerá na expressão.



The screenshot shows a code editor interface with a dark theme. At the top, there's a menu bar with options: File, Edit, Selection, View, Go, Run, Terminal, Help. Below the menu is a toolbar with icons for back, forward, and search. The main area is divided into two panes: 'EXPLORER' on the left and 'CODE' on the right. In the 'EXPLORER' pane, there's a tree view with 'OPEN EDITORS' (1 unsaved), 'CURSO-FRONTEND' expanded, showing 'app.js' and 'index.html'. In the 'CODE' pane, there's a code editor with the following content:

```
1 // ordem das operações matemáticas  
2 const result = 12 * (30 + 5) ** 3  
3  
4  
5  
6
```

The line 'const result = 12 * (30 + 5) ** 3' is highlighted with a red rectangle.

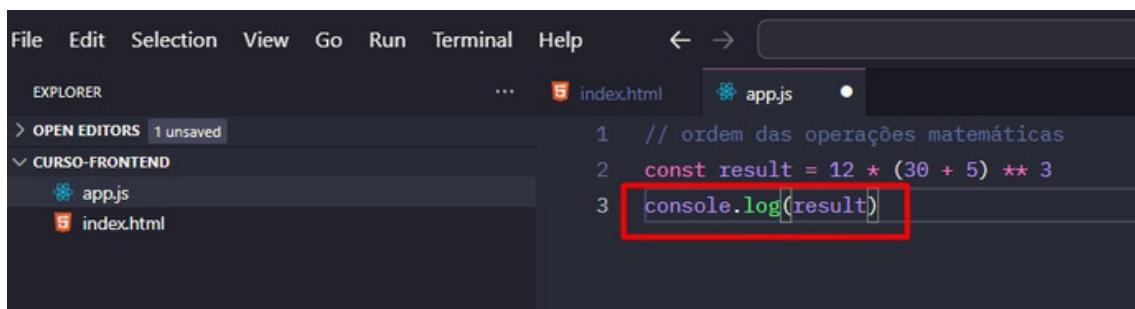


Primeiro tipo de dado: Numbers

Antes de qualquer coisa a expressão entre parênteses será resolvida, onde $30 + 5$ é igual a 35 e logo depois dos parênteses a expressão que envolve as potências é resolvida onde 35 elevado ao cubo é igual a 42875 e depois das potências a expressão que envolve multiplicação é resolvida, onde 12 vezes 42875 é igual a 514.500.

Logo abaixo vamos digitar `console.log` e passar a `result`, ficando assim:

```
console.log(result)
```

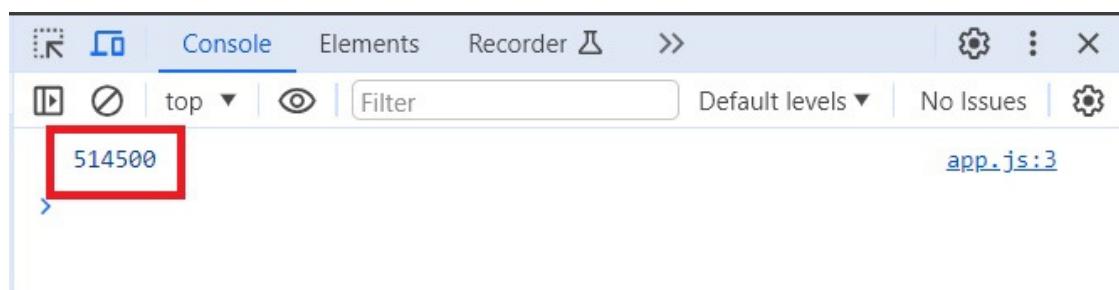


A screenshot of the Visual Studio Code interface. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The Explorer sidebar shows an open editor for 'index.html' and a file named 'app.js'. The code editor contains the following JavaScript code:

```
// ordem das operações matemáticas
const result = 12 * (30 + 5) ** 3
console.log(result)
```

The line `console.log(result)` is highlighted with a red rectangle.

E ao salvarmos o arquivo `app.js` com aquele `CTRL + S` maroto podemos ver no console do Navegador(Chrome) que o valor 514.500 será exibido no console



A screenshot of the Chrome DevTools Console tab. The tab bar includes Console, Elements, Recorder, and a gear icon. The main area shows the value `514500` with a red rectangle around it. The status bar at the bottom right indicates `app.js:3`.



Sugestões de prática:

1. Refazer e Entender os exemplos dessa aula;
2. Na página 7 dessa aula eu deixei **blocos de códigos** destacado tentar fazer o comentário do bloco de códigos dentro do VSCode
3. Usar valores definidos por vocês para praticar mais sobre: Módulos e ordem das operações matemáticas
Colocar isso em prática no VSCode e ver o resultado.

Programação é prática, curiosidade e repetição!!!!



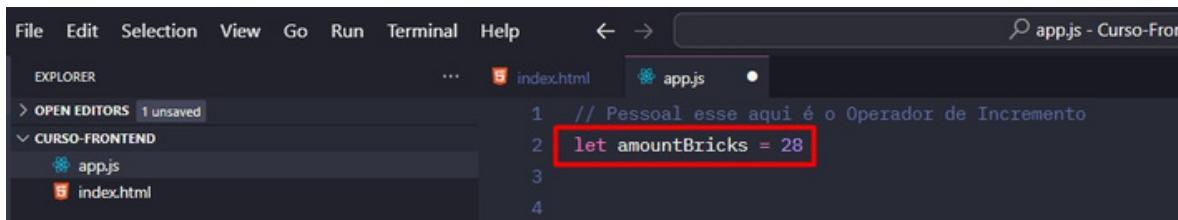
Numbers - Parte Final

Bora para a parte final dos Numbers, agora veremos os Ilustríssimos Operadores de Incremento e Decremento.

E para iniciarmos vamos até o Nosso arquivo app.js, declarar um comentário // Pessoal esse aqui é o Operador de Incremento

E abaixo desse comentário, bora declarar uma let amountBricks que recebe 28 (let amountBricks = 28). Ficando assim:

```
// Pessoal esse aqui é o Operador de Incremento  
let amountBricks = 28
```



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-Frontend  
EXPLORER ... index.html app.js ●  
OPEN EDITORS 1 unsaved  
CURSO-FRONTEND  
app.js  
index.html
```

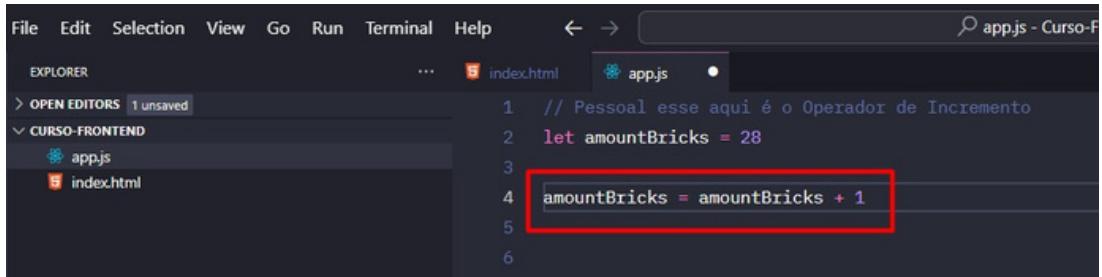
```
1 // Pessoal esse aqui é o Operador de Incremento  
2 let amountBricks = 28  
3  
4
```

Imagina que isso é o número de tijolos (amountBricks) que uma parede de uma casa tem e se quisermos adicionar + 1 tijolo para essa let. Podemos fazer desse jeito, especificaremos que amountBricks recebe amountBricks + 1 (amountBricks = amountBricks + 1) ou seja amountBricks recebe o valor que ela já tem + 1. Ficando assim:

```
amountBricks = amountBricks + 1
```



Numbers - Parte Final



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-Frontend
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui é o Operador de Incremento
2 let amountBricks = 28
3
4 amountBricks = amountBricks + 1
5
6
```

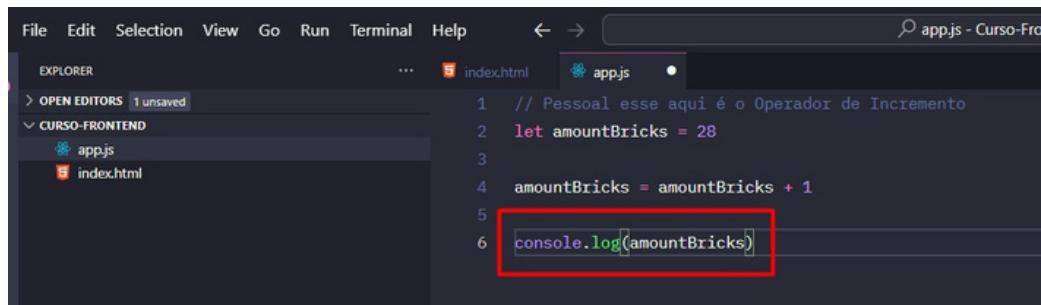
The screenshot shows a code editor window in VS Code. The file 'app.js' is open, containing the following code:

```
// Pessoal esse aqui é o Operador de Incremento
let amountBricks = 28
amountBricks = amountBricks + 1
```

The line 'amountBricks = amountBricks + 1' is highlighted with a red rectangle.

Adicionaremos um console.log passando a amountBricks. Ficando assim:

```
console.log(amountBricks)
```



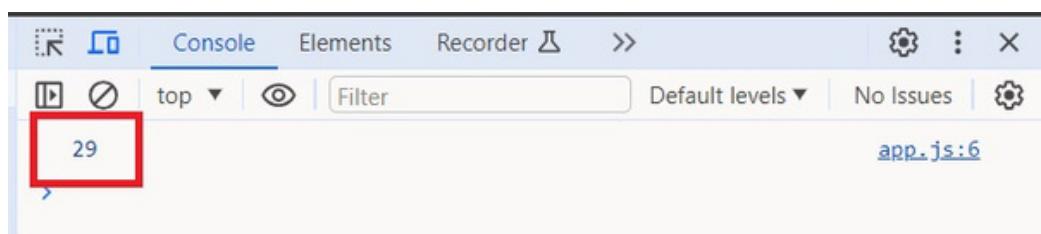
```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-Frontend
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui é o Operador de Incremento
2 let amountBricks = 28
3
4 amountBricks = amountBricks + 1
5
6 console.log(amountBricks)
```

The screenshot shows a code editor window in VS Code. The file 'app.js' is open, containing the following code:

```
// Pessoal esse aqui é o Operador de Incremento
let amountBricks = 28
amountBricks = amountBricks + 1
console.log(amountBricks)
```

The line 'console.log(amountBricks)' is highlighted with a red rectangle.

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o número 29 é exibido.

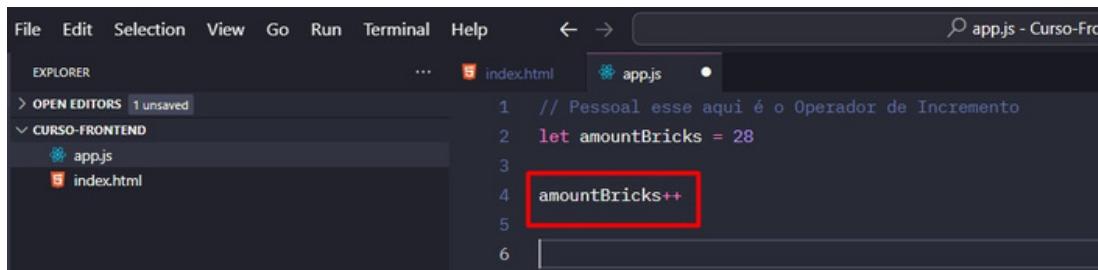


/igor-rebolla

Numbers - Parte Final

Nas páginas 2 e 3 dessa aula vimos uma forma mais didática de como fazer isso (Pois caso vocês encontrem essa forma em algum projeto que estejam atuando, já sabem que é possível fazer assim também). Só que existe um jeitinho maroto (que hoje em dia nos projetos é o que é mais utilizado) pra representarmos essa mesma expressão e chegamos ao mesmo resultado. Chamamos de "shorthand" (atalho ou forma abreviada).

amountBricks++



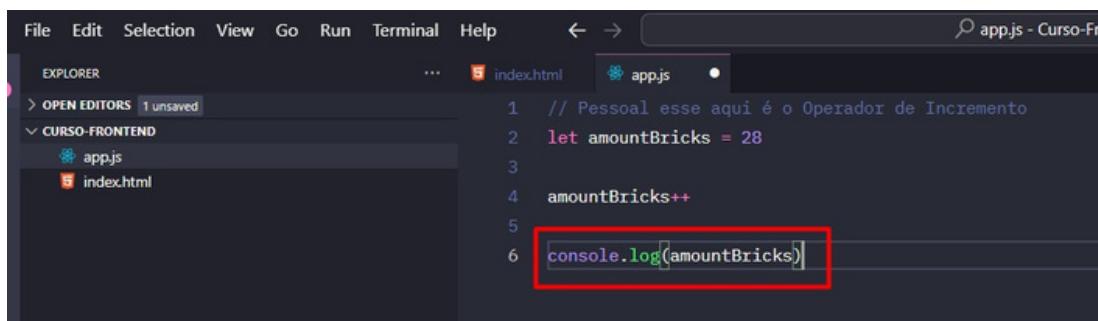
A screenshot of a code editor showing a file named 'app.js'. The code contains the following lines:

```
// Pessoal esse aqui é o Operador de Incremento
let amountBricks = 28
amountBricks++
```

The line 'amountBricks++' is highlighted with a red box.

Adicionaremos um `console.log` passando a `amountBricks`. Ficando assim:

`console.log(amountBricks)`



A screenshot of a code editor showing the same 'app.js' file with the following code:

```
// Pessoal esse aqui é o Operador de Incremento
let amountBricks = 28
amountBricks++
console.log(amountBricks)
```

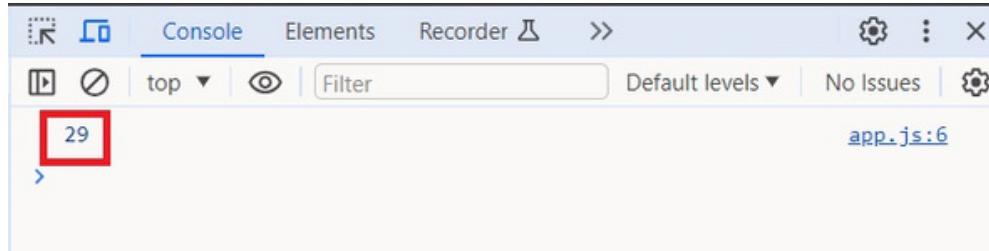
The line `console.log(amountBricks)` is highlighted with a red box.



/igor-rebolla

Numbers - Parte Final

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o número 29 é exibido.



O nome do que acabamos de fazer agora é o incremento, nós incrementamos em 1 o valor da variável ou seja fizemos com que ela recebesse todo o valor que ela já tinha + 1.

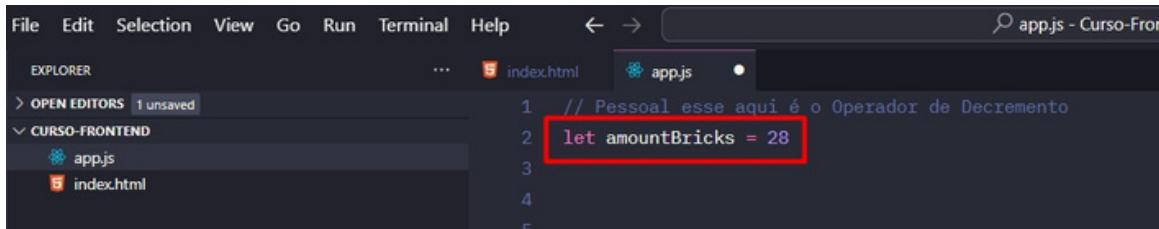


Numbers - Parte Final

Dentro do nosso arquivo app.js, bora declarar um comentário: // Pessoal esse aqui é o Operador de Decremento

E abaixo desse comentário, bora declarar uma let amountBricks que recebe 28 (let amountBricks = 28). Ficando assim:

```
// Pessoal esse aqui é o Operador de Decremento  
let amountBricks = 28
```



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-Frontend  
EXPLORER ... index.html app.js ●  
> OPEN EDITORS 1 unsaved  
CURSO-FRONTEND  
app.js  
index.html
```

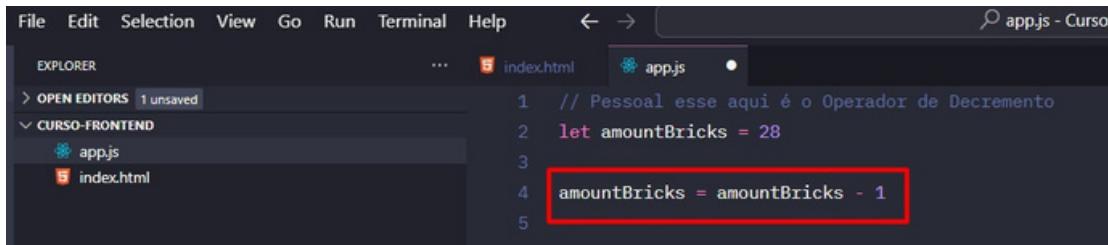
The screenshot shows a code editor interface with a dark theme. At the top, there's a menu bar with options like File, Edit, Selection, View, Go, Run, Terminal, and Help. Below the menu is a toolbar with navigation icons. The main area is titled "app.js - Curso-Frontend". On the left, there's an "EXPLORER" sidebar showing files "index.html" and "app.js" under a folder "CURSO-FRONTEND". The "OPEN EDITORS" section indicates "1 unsaved". The code editor itself has two lines of code: line 1 contains the multi-line comment "// Pessoal esse aqui é o Operador de Decremento", and line 2 contains the declaration "let amountBricks = 28". The line "let amountBricks = 28" is highlighted with a red rectangular box.

Imagina que isso é o número de tijolos (amountBricks) que uma parede de uma casa tem e se quisermos diminuir 1 tijolo para essa let (porque o nosso orçamento da curto e teremos que tirar um tijolo da nossa lista). Nós podemos desse jeito, especificaremos que amountBricks recebe amountBricks - 1 (amountBricks = amountBricks - 1) ou seja amountBricks recebe o valor que ela já tem - 1. Ficando assim:

```
amountBricks = amountBricks - 1
```



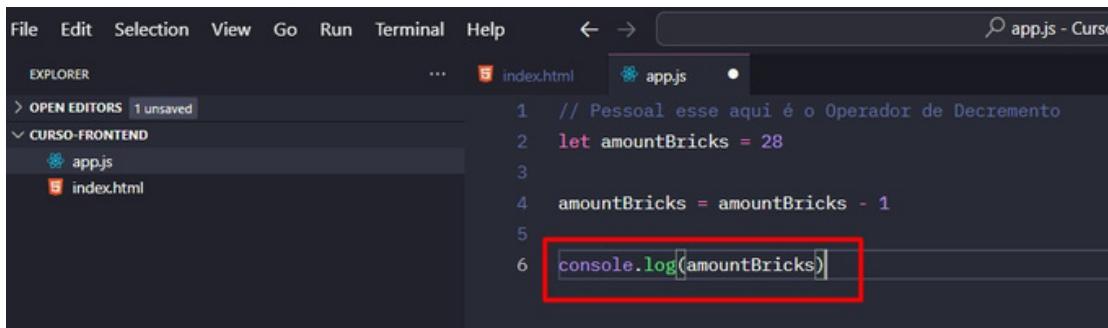
Numbers - Parte Final



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso
EXPLORER ... index.html app.js
> OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui é o Operador de Decremento
2 let amountBricks = 28
3
4 amountBricks = amountBricks - 1
5
```

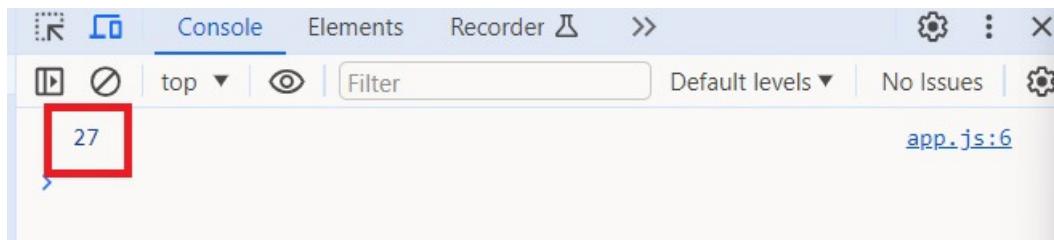
Adicionaremos um `console.log` passando a `amountBricks`. Ficando assim:

```
console.log(amountBricks)
```



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso
EXPLORER ... index.html app.js
> OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui é o Operador de Decremento
2 let amountBricks = 28
3
4 amountBricks = amountBricks - 1
5
6 console.log(amountBricks)
```

E ao salvarmos o arquivo `app.js` com aquele `CTRL + S` maroto podemos ver no console do Navegador(Chrome) que o número 27 é exibido.



```
Console Elements Recorder > ⚙️ ⓘ app.js:6
27
```

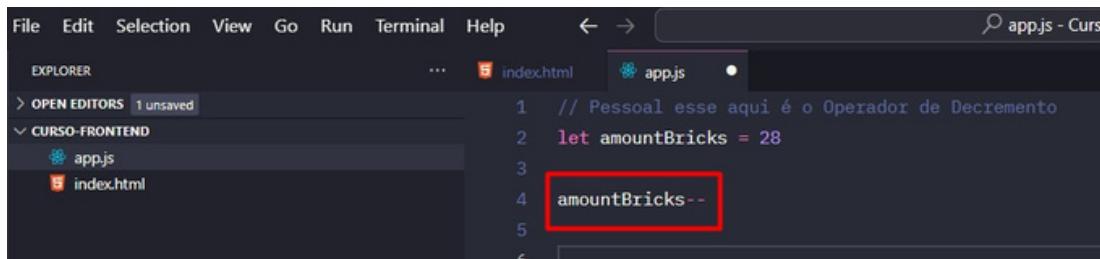


/igor-rebolla

Numbers - Parte Final

Assim como existe um "shorthand" (atalho ou forma abreviada) pra representarmos essa mesma expressão e no final chegarmos ao mesmo resultado.

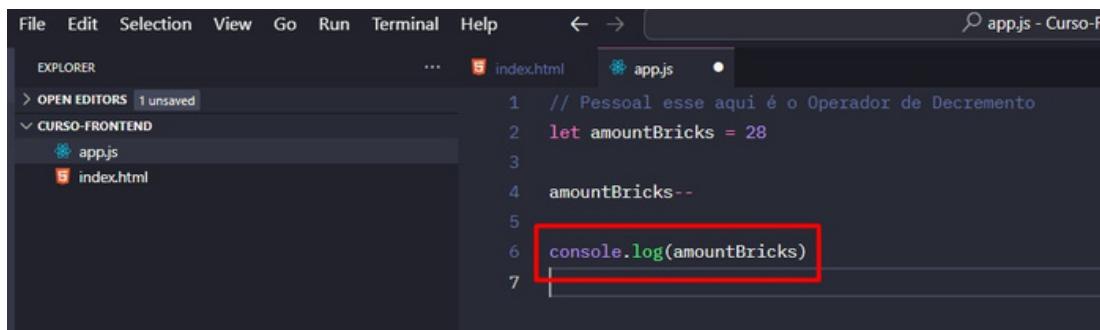
amountBricks--



```
File Edit Selection View Go Run Terminal Help ⌘ ⌘ app.js - Curs
EXPLORER ...
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui é o Operador de Decremento
2 let amountBricks = 28
3
4 amountBricks--
```

Adicionaremos um console.log passando a amountBricks. Ficando assim:

console.log(amountBricks)



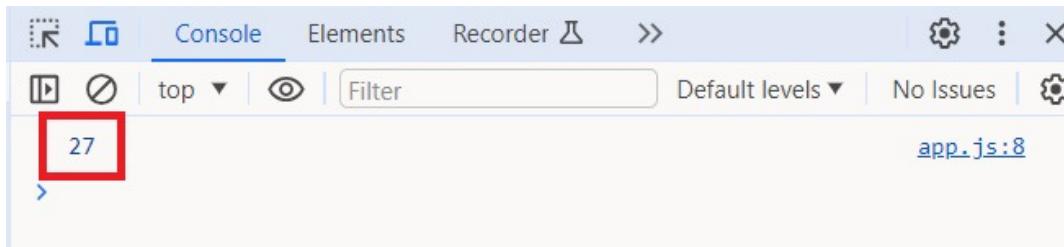
```
File Edit Selection View Go Run Terminal Help ⌘ ⌘ app.js - Curso-F
EXPLORER ...
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui é o Operador de Decremento
2 let amountBricks = 28
3
4 amountBricks--
5
6 console.log(amountBricks)
7 |
```



/igor-rebolla

Numbers - Parte Final

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o número 27 é exibido.



O nome do que acabamos de fazer é o decremento, nós decrementamos em 1 o valor da variável ou seja fizemos com que ela recebesse todo o valor que ela já tinha - 1.

Numbers - Parte Final

Addition Assignment

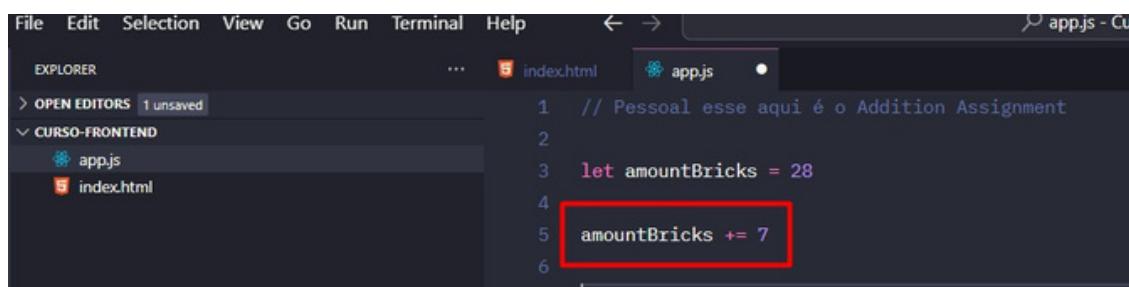
Igor, incrementamos 1 e decrementamos 1 nos exemplos anteriores, e se eu quiser adicionar mais de 1... por exemplo se eu quiser adicionar + 7 ?
Ótima pergunta, essa é a missão para o nosso amigo: O Operador Addition Assignment.

Bora ver esse tal de Addition Assignment....

Dentro do nosso arquivo app.js, bora declarar um comentário: // Pessoal esse aqui é o Addition Assignment

E abaixo desse comentário, usaremos o número 7 (assim como o questionamento da pergunta acima) e faremos com que a amountBricks receba amountBricks += 7 (ou seja o valor que ela já tem + 7). Ficando assim:

```
amountBricks += 7
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved
CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui é o Addition Assignment
2
3 let amountBricks = 28
4
5 amountBricks += 7
```

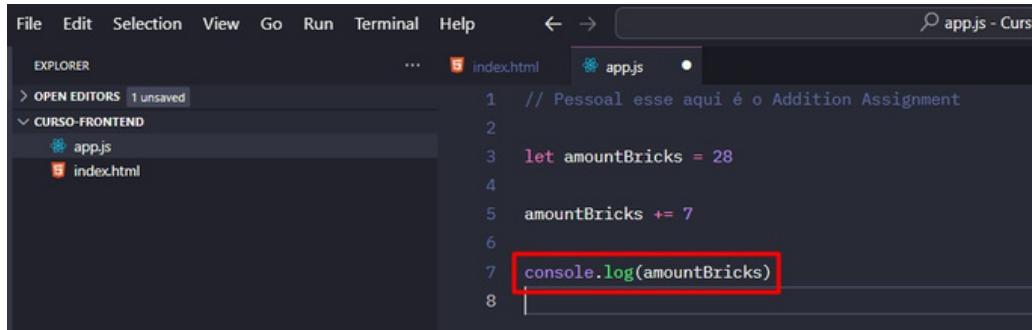
Adicionaremos um console.log passando a amountBricks. Ficando assim:

```
console.log(amountBricks)
```



Numbers - Parte Final

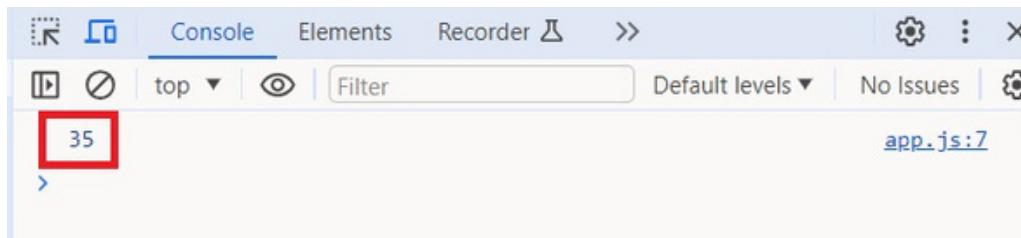
Addition Assignment



```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html

1 // Pessoal esse aqui é o Addition Assignment
2
3 let amountBricks = 28
4
5 amountBricks += 7
6
7 console.log(amountBricks)
8
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o número 35 é exibido.



```
Console Elements Recorder >
top Filter Default levels No Issues app.js:7
35
```



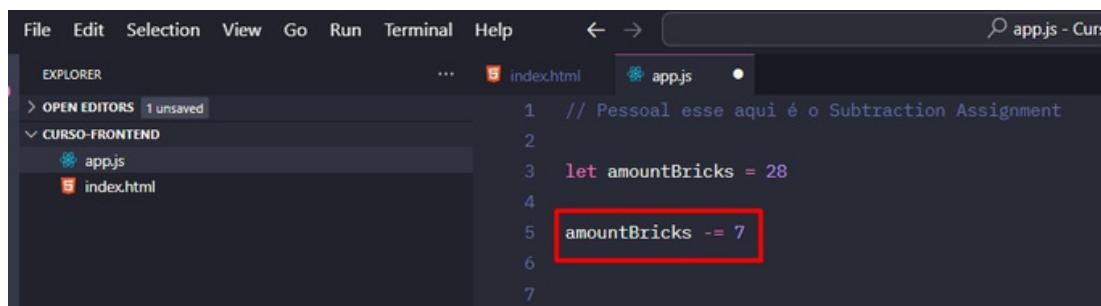
Numbers - Parte Final Subtraction Assignment

Bora ver agora esse tal de Subtraction Assignment....

Dentro do nosso arquivo app.js, bora declarar um comentário: // Pessoal esse aqui é o Subtraction Assignment

E abaixo desse comentário, usaremos o número 7 (assim como o questionamento da pergunta feita antes de estudarmos sobre o Addition Assignment) e faremos com que a amountBricks receba amountBricks -= 7 (ou seja o valor que ela já tem - 7). Ficando assim:

amountBricks -= 7



The screenshot shows a code editor interface with a dark theme. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar says 'app.js - Curs'. The left sidebar shows an 'EXPLORER' section with 'OPEN EDITORS 1 unsaved' and a 'CURSO-FRONTEND' folder containing 'app.js' and 'index.html'. The main code editor area has the following content:

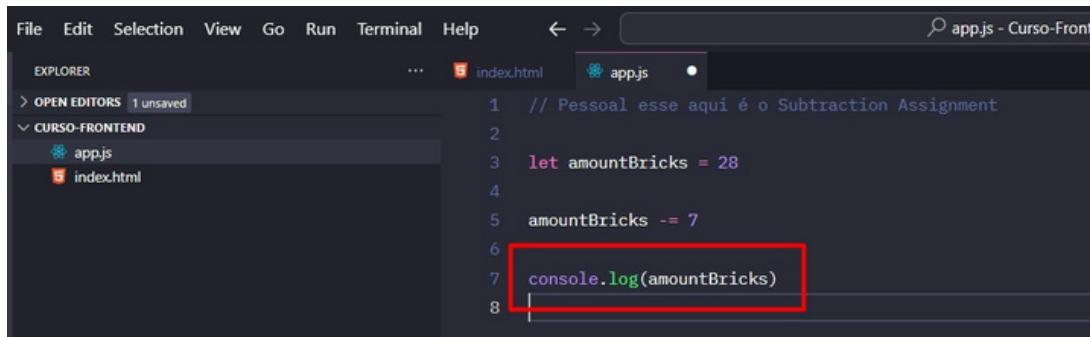
```
// Pessoal esse aqui é o Subtraction Assignment
let amountBricks = 28
amountBricks -= 7
```

Adicionaremos um console.log passando a amountBricks. Ficando assim:

console.log(amountBricks)



Numbers - Parte Final Subtraction Assignment

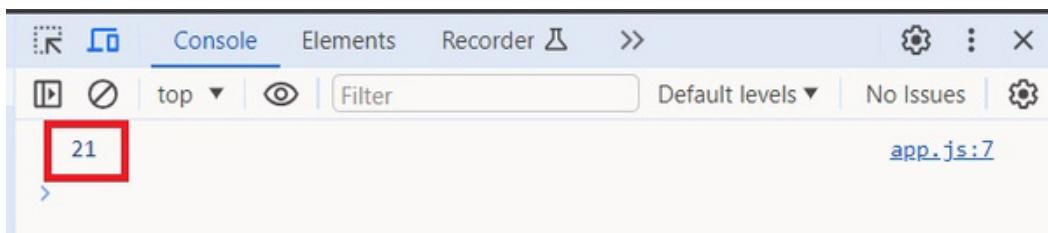


A screenshot of the Visual Studio Code interface. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar shows "app.js - Curso-Frontend". The Explorer sidebar shows "OPEN EDITORS 1 unsaved" and "CURSO-FRONTEND" with files "app.js" and "index.html". The main editor area contains the following code:

```
1 // Pessoal esse aqui é o Subtraction Assignment
2
3 let amountBricks = 28
4
5 amountBricks -= 7
6
7 console.log(amountBricks)
8
```

The line "console.log(amountBricks)" is highlighted with a red box.

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o número 21 é exibido.



Numbers - Parte Final

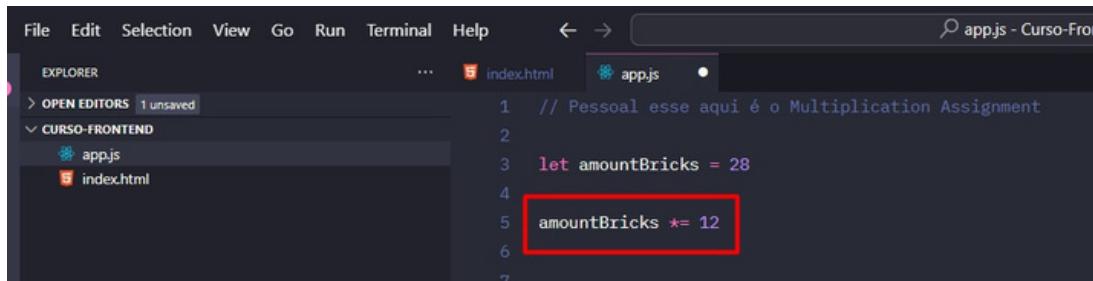
Multiplication Assignment

Bora ver agora esse tal de Multiplication Assignment....

Dentro do nosso arquivo app.js, bora declarar um comentário: // Pessoal esse aqui é o Multiplication Assignment

E abaixo desse comentário, usaremos o número 12 e faremos com que a amountBricks receba amountBricks *= 7 (ou seja o valor que ela já tem multiplicado por 12). Ficando assim:

```
amountBricks *= 12
```



The screenshot shows the VS Code interface with the 'app.js' file open. The code is as follows:

```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-Frontend
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui é o Multiplication Assignment
2
3 let amountBricks = 28
4
5 amountBricks *= 12
6
7
```

The line 'amountBricks *= 12' is highlighted with a red rectangle.

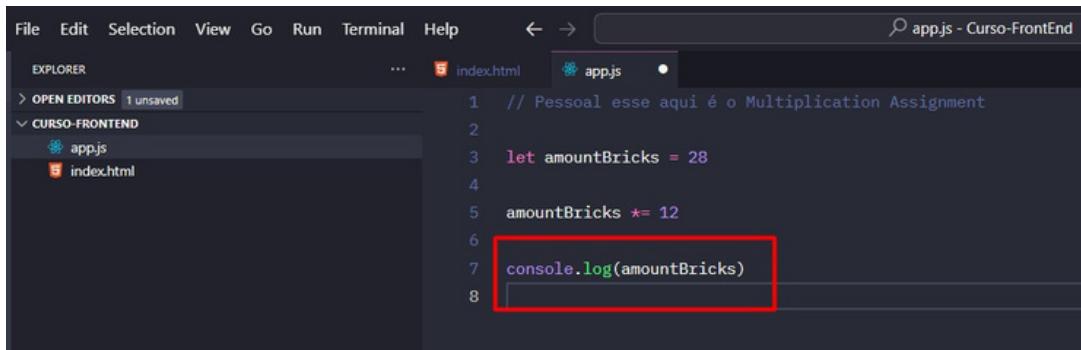
OBS.: Só relembrando que o sinal de multiplicação dentro do JavaScript(JS) é representado pelo asterisco (*).

Adicionaremos um console.log passando a amountBricks. Ficando assim:

```
console.log(amountBricks)
```

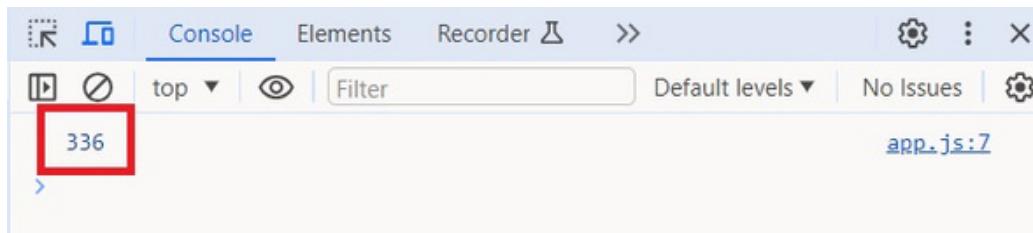


Numbers - Parte Final Multiplication Assignment



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER ... 5 index.html app.js
> OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui é o Multiplication Assignment
2
3 let amountBricks = 28
4
5 amountBricks *= 12
6
7 console.log(amountBricks)
8
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o número 336 é exibido.



```
Console Elements Recorder > ⚙️ ⚙️ X
top Filter Default levels No Issues ⚙️
336 app.js:7
```



Numbers - Parte Final

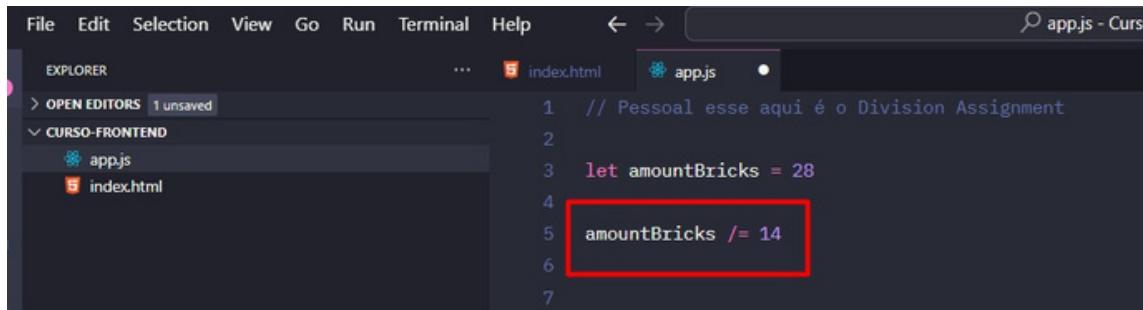
Division Assignment

Bora ver agora esse tal de Division Assignment....

Dentro do nosso arquivo app.js, bora declarar um comentário: // Pessoal esse aqui é o Division Assignment

E abaixo desse comentário, usaremos o número 14 e faremos com que a amountBricks receba amountBricks /= 14 (ou seja o valor que ela já tem dividido por 14). Ficando assim:

amountBricks *= 14



A screenshot of a code editor showing the file app.js. The code contains the following lines:

```
1 // Pessoal esse aqui é o Division Assignment
2
3 let amountBricks = 28
4
5 amountBricks /= 14
6
7
```

The line `amountBricks /= 14` is highlighted with a red rectangle.

OBS.: Só relembrando que o sinal de divisão dentro do JavaScript(JS) é representado pelo barra (/).

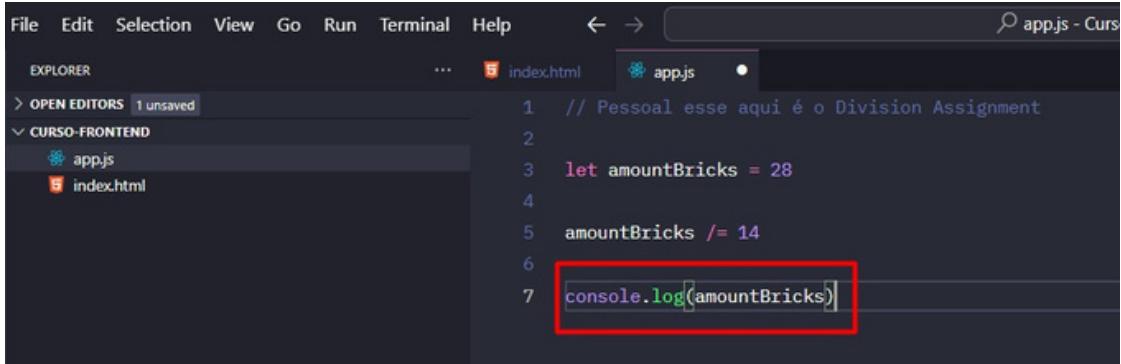
Adicionaremos um console.log passando a amountBricks. Ficando assim:

`console.log(amountBricks)`



Numbers - Parte Final

Division Assignment

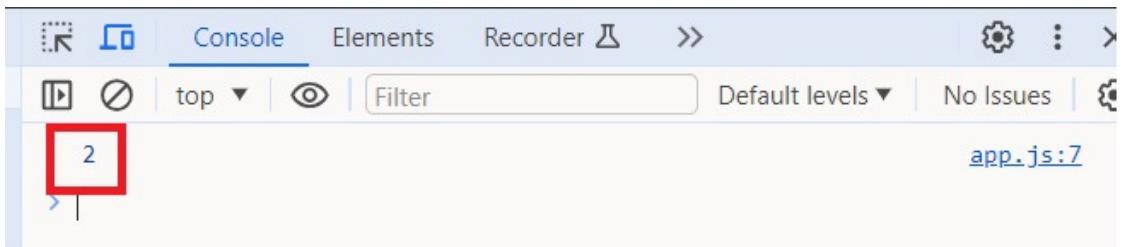


A screenshot of a code editor showing the file `app.js`. The code contains the following JavaScript:

```
// Pessoal esse aqui é o Division Assignment
let amountBricks = 28
amountBricks /= 14
console.log(amountBricks)
```

The line `console.log(amountBricks)` is highlighted with a red box.

E ao salvarmos o arquivo `app.js` com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o número 2 é exibido.



A screenshot of the Chrome DevTools Console tab. The number `2` is displayed in the console, with a red box highlighting it.



Numbers - Parte Final

NaN - Not a Number

Bora ver agora esse tal de NaN – Not a Number....

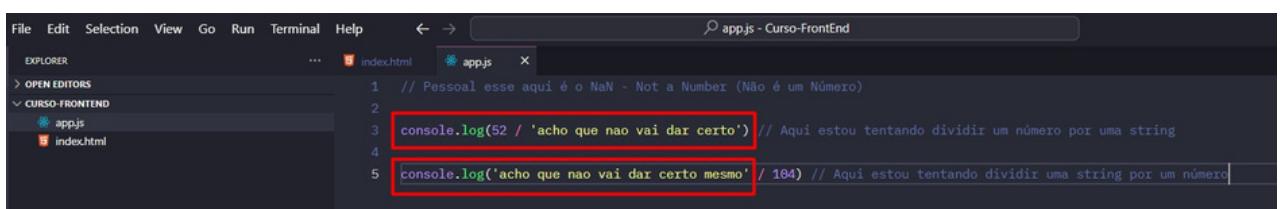
Dentro do nosso arquivo app.js, bora declarar um comentário: // Pessoal esse aqui é o NaN – Not a Number (Não é um Número)

Veremos esse valor quando tentarmos fazer algum tipo de operação que não faça nenhum sentido, ou seja algum tipo de operação que no final não venha resultar em um número

Aqui abaixo declararemos dois consoles.log.

O primeiro tentaremos dividir um número por uma string. Ficando assim:
console.log(52 / 'acho que nao vai dar certo')

O segundo tentaremos dividir uma string por um número. Ficando assim:
console.log('acho que nao vai dar certo mesmo' / 104)



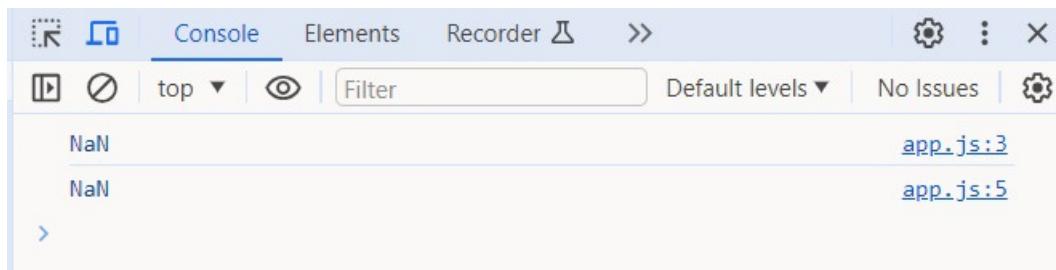
```
File Edit Selection View Go Run Terminal Help ↵ → 🔍 app.js - Curso-FrontEnd
EXPLORER ... index.html app.js ✘
> OPEN EDITORS
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui é o NaN - Not a Number (Não é um Número)
2
3 console.log(52 / 'acho que nao vai dar certo') // Aqui estou tentando dividir um número por uma string
4
5 console.log('acho que nao vai dar certo mesmo' / 104) // Aqui estou tentando dividir uma string por um número
```



Numbers - Parte Final

NaN - Not a Number

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que os dois NaN (Not a Number) foram exibidos um embaixo do outro.



E isso ocorreu porque tentamos fazer uma operação que não resulta em um número. Então se você receber esse valor no momento em que você tiver programando é porque existiu naquele exato momento uma tentativa de executar uma operação que não fez nenhum sentido.

E não se preocupe com isso agora (pode ter ficado meio confuso), aqui é para mostrar os conceitos e para que vocês tenham o primeiro contato com esse carinha o NaN que parece "birutão" no inicio, mas é brother. No decorrer do curso, veremos mais exemplos do NaN.

Strings - Segundo Tipo de Dados

O Segundo tipo de dados que exploraremos com um pouquinho mais de detalhes agora, são as STIRINGS:

Igor, uma dúvida.... Por que nós usamos Strings?

Boa pergunta: Pra armazenar (Letras, números ou qualquer outro tipo de caractere) nós podemos usar uma string para armazenar um e-mail por ou um nome de uma rua por exemplo

A primeira coisa que faremos aqui no app.js é exibir uma string no console. Então a especificaremos um console.log() e é importantíssimo lembrarmos disso: Armazenamos strings utilizando ou aspas simples ou aspas duplas, independente de qual você optar o comportamento será o mesmo, mas se você optar abrir a declaração de uma string com aspas simples por exemplo devemos fechar a string também com aspas simples e o mesmo vale também para strings com aspas duplas. Então aqui dentro dos parênteses do console.log e dentro das aspas simples nesse caso vou escrever um 'Aula 19'. Ficando assim:

```
// Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
```

```
console.log('Aula 19')
```



The screenshot shows a code editor interface with the following details:

- File menu: File, Edit, Selection, View, Go, Run, Terminal, Help.
- Toolbar: Back, Forward, Search icon, and a tab labeled "app.js - Curso-FrontEnd".
- Explorer sidebar: Shows "OPEN EDITORS 1 unsaved" and a "CURSO-FRONTEND" folder containing "app.js" and "index.html".
- Code editor area:

```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2
3 console.log('Aula 19')
4
5
```

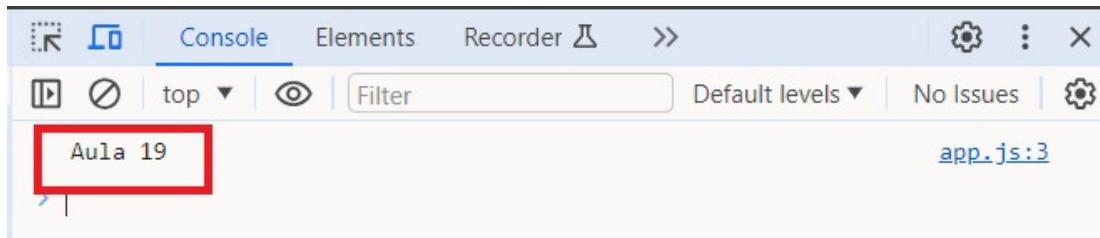
The line "console.log('Aula 19')" is highlighted with a red rectangular box.



/igor-rebolla

Strings - Segundo Tipo de Dados

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que a string Aula 19 é exibido.



Igor, Igor, Igor..... e as aspas onde foram parar que não foram exibidas no console?

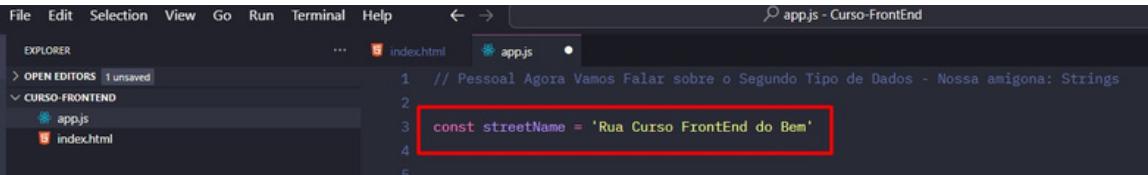
As nobres aspas aparecem apenas dentro do VSCode para que o mesmo consiga identificar que ali é uma string, e a partir do momento que é renderizado/mostrado no navegador as aspas não são mais necessárias mostrando apenas o conteúdo dentro dela (isso vale tanto para aspas duplas quanto para aspas simples).

Podemos armazenar também uma string dentro de uma variável, então abaixo do console.log, declararemos uma const streetName que vai receber uma string 'Rua Curso FrontEnd do Bem', então agora nós temos um nome, armazenado na const streetName. Ficando assim:

```
const streetName = 'Rua Curso FrontEnd do Bem'
```



Strings - Segundo Tipo de Dados



```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
index.html
app.js
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2
3 const streetName = 'Rua Curso FrontEnd do Bem'
4
```

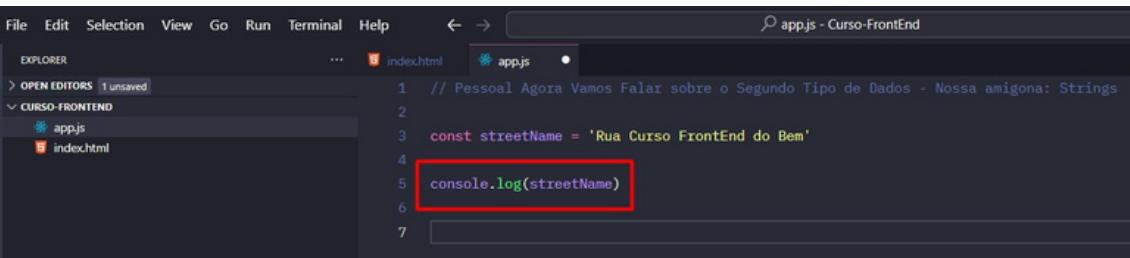
A screenshot of the Visual Studio Code interface. The title bar says "app.js - Curso-FrontEnd". The left sidebar shows an "EXPLORER" section with "OPEN EDITORS 1 unsaved" and a "CURSO-FRONTEND" folder containing "app.js" and "index.html". The main editor area has the following code:

```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2
3 const streetName = 'Rua Curso FrontEnd do Bem'
4
```

The line "const streetName = 'Rua Curso FrontEnd do Bem'" is highlighted with a red rectangle.

Podemos exibir essa constante no console.log passando a const streetName entre parênteses. Ficando assim:

```
console.log(streetName)
```



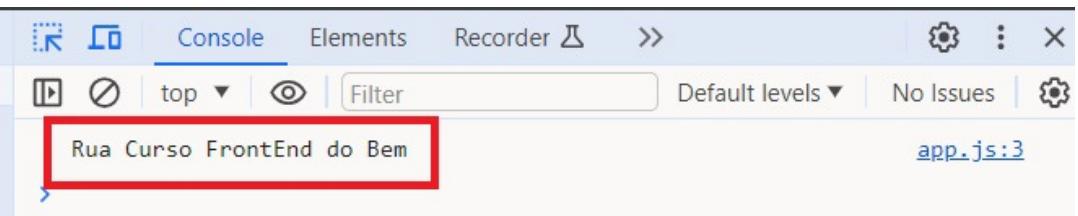
```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
index.html
app.js
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2
3 const streetName = 'Rua Curso FrontEnd do Bem'
4
5 console.log(streetName)
6
7
```

A screenshot of the Visual Studio Code interface, similar to the previous one but with a new line of code added. The main editor area now contains:

```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2
3 const streetName = 'Rua Curso FrontEnd do Bem'
4
5 console.log(streetName)
6
7
```

The line "console.log(streetName)" is highlighted with a red rectangle.

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que a string Rua Curso FrontEnd do Bem é exibido no console.



```
Console Elements Recorder > Settings More X
top Filter Default levels No Issues
Rua Curso FrontEnd do Bem app.js:3
```

A screenshot of the Chrome DevTools "Console" tab. The tab bar has "Console" selected. The console output shows the string "Rua Curso FrontEnd do Bem" followed by the file name "app.js:3". The entire output line is highlighted with a red rectangle.

/igor-rebolla

Strings

Concatenação de Strings

Bora ver agora essa tal de Concatenação de Strings....

Imaginaremos que temos 2 strings e gostaríamos de juntar essas duas strings, no fascinante mundo da computação chamamos isso de concatenação. Concatenação é o nome técnico usado em momentos onde uma coisa se junta a outra (ou seja sabe a fusão do Goku em Dragon Ball, é a mesma coisa).

Dentro do nosso arquivo app.js, bora declarar dois comentários:

```
// Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings  
// Concatenação de Strings
```

Bora então imaginar que nós temos 2 consts. Declararemos uma const houseDogName que recebe uma string 'Catiorro' e na linha abaixo declararemos uma segunda const houseCatName que recebe uma string 'Bichano'. Ficando assim:

```
const houseDogName = 'Catiorro'  
const houseCatName = 'Bichano'
```



The screenshot shows a code editor interface with the following details:

- File menu: File, Edit, Selection, View, Go, Run, Terminal, Help.
- Toolbar: Back, Forward, Search bar (app.js - Curso-FrontEnd).
- Explorer sidebar: OPEN EDITORS (index.html, app.js), CURSO-FRONTEND (app.js, index.html).
- Code area:

```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Concatenação de Stings
3
4 const houseDogName = 'Catiorro'
5 const houseCatName = 'Bichano'
```

The lines from 4 to 5 are highlighted with a red rectangle.



/igor-rebolla

Strings

Concatenação de Strings

E bora imagina novamente que a gostaríamos de juntar essas duas strings, então declararemos uma nova const houseAnimalsNames que recebe houseDogName concatenada com houseCatName, isso é uma concatenação de strings, nós usamos um sinal de + pra concatenar uma string a outra (em outras palavras fazer a fusão do Goku). Ficando assim:

```
const houseAnimalsNames = houseDogName + houseCatName
```



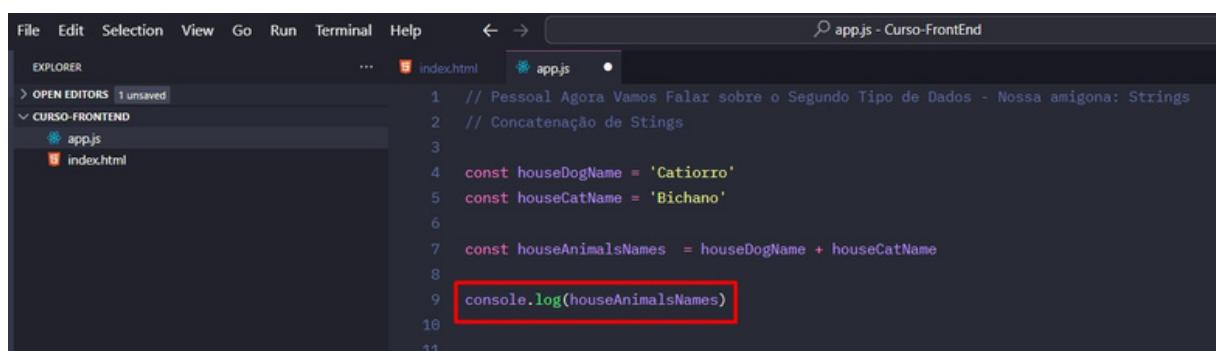
A screenshot of the Visual Studio Code interface. The title bar says "app.js - Curso-FrontEnd". The left sidebar shows an "EXPLORER" section with "OPEN EDITORS 1 unsaved" and a "CURSO-FRONTEND" folder containing "app.js" and "index.html". The main editor area has the following code:

```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Concatenação de Stings
3
4 const houseDogName = 'Catiorro'
5 const houseCatName = 'Bichano'
6
7 const houseAnimalsNames = houseDogName + houseCatName
8
9
```

The line "const houseAnimalsNames = houseDogName + houseCatName" is highlighted with a red rectangle.

Adicionaremos um console.log passando a houseAnimalsNames . Ficando assim:

```
console.log(houseAnimalsNames )
```



A screenshot of the Visual Studio Code interface, identical to the previous one but with an additional line of code added at the end:

```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Concatenação de Stings
3
4 const houseDogName = 'Catiorro'
5 const houseCatName = 'Bichano'
6
7 const houseAnimalsNames = houseDogName + houseCatName
8
9 console.log(houseAnimalsNames)
10
11
```

The line "console.log(houseAnimalsNames)" is highlighted with a red rectangle.

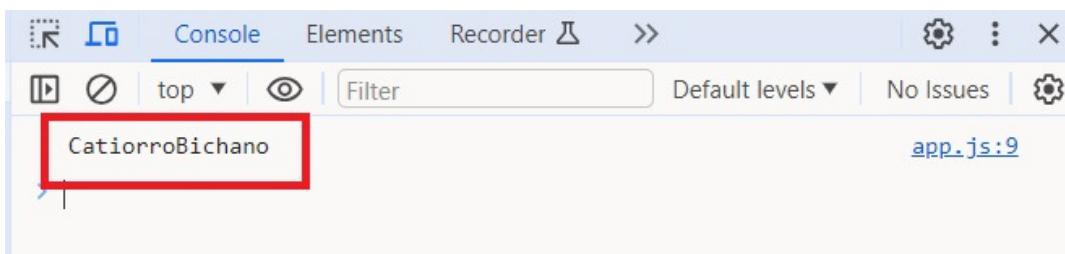


/igor-rebolla

Strings

Concatenação de Strings

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que as duas strings concatenadas são exibidas no console.



Notem que aqui não existe um espaço entre as strings (ficou tudo grudado... “sai pra lá bichano”, “sai você pra lá catiorro”... calma meus amiguinhos não briguem já resolvemos isso).

Para conseguirmos espaçar esse 2 nomes, logo depois do sinal de + podemos especificar uma string '' que tem um espaço em branco e depois desse espaço em branco especificamos outro +.

Igor, Igor.... O que aconteceu aqui, ficou meio confuso?

Nós estamos concatenando 3 coisas diferentes (houseDogName, uma string que contém um espaço em branco e o houseCatName). Ficando assim:

A screenshot of a code editor showing the 'app.js' file. The file contains the following code:

```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Concatenação de Strings
3
4 const houseDogName = 'Catiorro'
5 const houseCatName = 'Bichano'
6
7 const houseAnimalsNames = houseDogName + ' ' + houseCatName
8
```

The line 'const houseAnimalsNames = houseDogName + ' ' + houseCatName' is highlighted with a red box.

/igor-rebolla

Strings

Concatenação de Strings

Adicionaremos um `console.log` passando a `houseAnimalsNames`. Ficando assim:

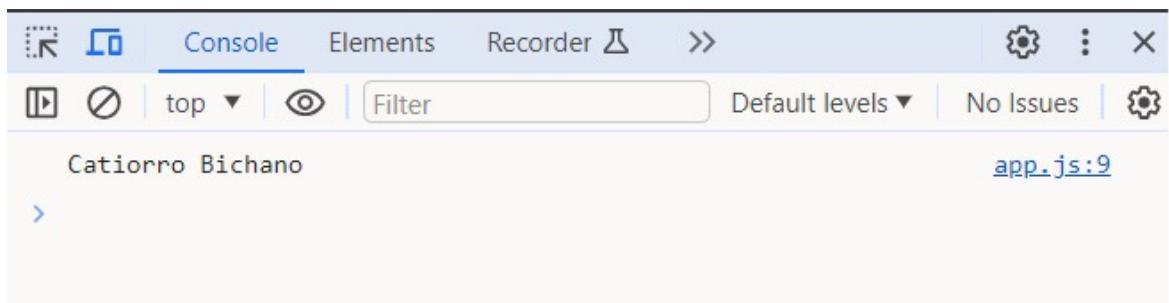
```
console.log(houseAnimalsNames )
```

The screenshot shows a code editor interface with a dark theme. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar says "app.js - Curso-FrontEnd". The left sidebar shows an "EXPLORER" section with "OPEN EDITORS 1 unsaved" and a tree view for "CURSO-FRONTEND" containing "app.js" and "index.html". The main code area has the following content:

```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Concatenação de Stings
3
4 const houseDogName = 'Catiorro'
5 const houseCatName = 'Bichano'
6
7 const houseAnimalsNames = houseDogName + ' ' + houseCatName
8
9 console.log(houseAnimalsNames)
10
```

The line `console.log(houseAnimalsNames)` is highlighted with a red rectangle.

E ao salvarmos o arquivo `app.js` com aquele `CTRL + S` maroto podemos ver no console do Navegador(Chrome) que agora existe um espaço entre as duas strings concatenadas.



/igor-rebolla

Strings

Como acessar Caracteres

Podemos acessar também um único caractere de uma string que nós armazenamos. E para fazermos isso declaremos um `console.log` e passaremos a `houseDogName`, só que gostaríamos de exibir, só o primeiro caractere (ou seja, a primeira letra) da `houseDogName` que no nosso exemplo aqui é o 'C' de Catiorro.

Podemos fazer isso abrindo e fechando colchetes no fim do nome da const `houseDogName[]` e especificarmos o número 0 entre esses colchetes [0] para assim acessarmos a primeira letra da `houseDogName`.

Igor, e porque esse número 0 e não o número 1 ali nos colchetes?

Bora Pergunta: Porque o JavaScript(JS) é uma linguagem baseada no Zero (Zero-Based). Isso significa que a contagem ao invés de iniciar em 1, vai começar a partir do 0), então em JavaScript(JS) esse letra 'C' está na posição 0, essa letra 'a' está na posição 1, essa letra 't' está na posição 2, essa letra 'i' está na posição 3, essa letra 'o' está na posição '4', essa letra 'r' está na posição 5, essa letra 'r' está na posição 6, essa letra 'o' está na posição 7. Então é por isso que especificamos essa notação de colchetes e número no fim da const, para acessarmos um caractere específico que está em uma determinada posição na string. Ficando assim:

```
console.log(houseDogName[0])
```



/igor-rebolla

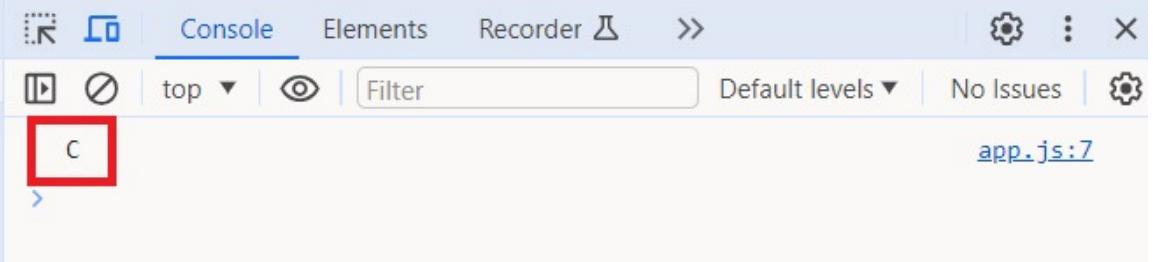
Strings

Como acessar Caracteres



```
File Edit Selection View Go Run Terminal Help ... index.html app.js
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Como Acessar o(s) Caracter(es)
3
4 const houseDogName = 'Catorro'
5 const houseCatName = 'Bichano'
6
7 console.log(houseDogName[0])
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o caractere C é exibido.



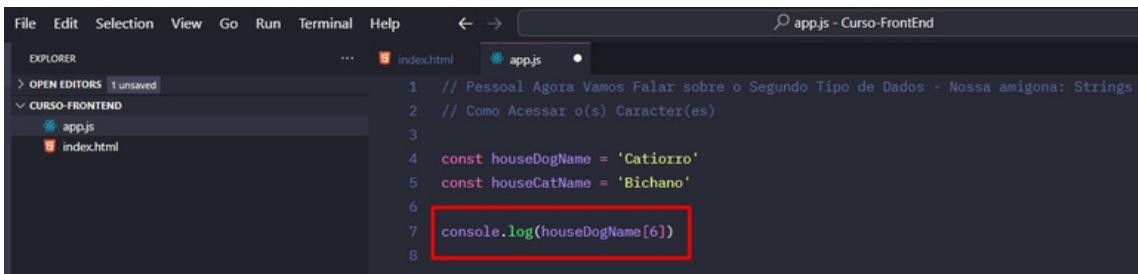
```
Console Elements Recorder > Default levels ▾ No Issues ↴
C
app.js:7
```

Se quisermos que o segundo 'r' seja exibido no console, mudamos o número 0 para 6, porque a posição 0 (corresponde ao C), a posição 1 (corresponde ao a), a posição 2 (corresponde ao t) e assim até chegar no segundo r cuja posição é a 6. Ficando assim:

```
console.log(houseDogName[6])
```

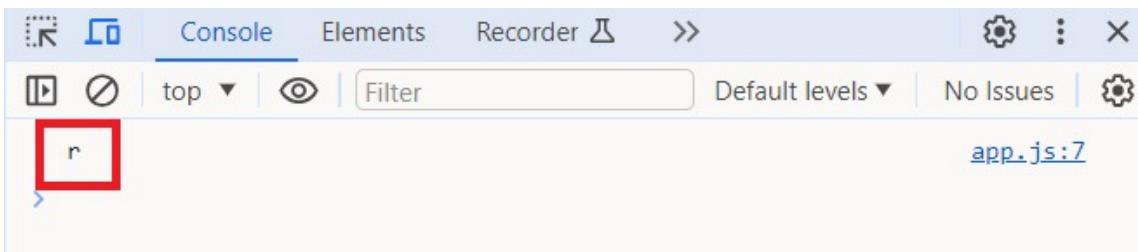
Strings

Como acessar Caracteres



```
File Edit Selection View Go Run Terminal Help ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Como Acessar o(s) Caracter(es)
3
4 const houseDogName = 'Catiorro'
5 const houseCatName = 'Bichano'
6
7 console.log(houseDogName[6])
8
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o segundo caractere r é exibido.



```
Console Elements Recorder > top Filter Default levels No Issues app.js:7
r
```

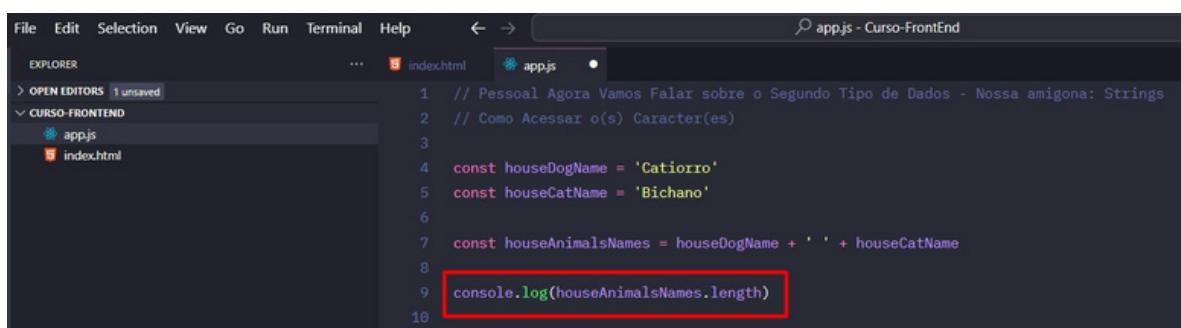


Strings

Comprimento de uma String

As nossas amigas Strings também possuem propriedades e métodos. E uma das propriedades que uma string conta é o length, o length é uma propriedade que vai nos informar o comprimento de uma string ou seja quantos caracteres ela tem. Declaremos um console.log que vai exibir houseAnimalsNames e para conseguirmos especificar uma propriedade digitaremos ponto(.) e em seguida o nome da propriedade aqui no nosso exemplo é a length. Ficando assim:

```
console.log(houseAnimalsNames.length)
```

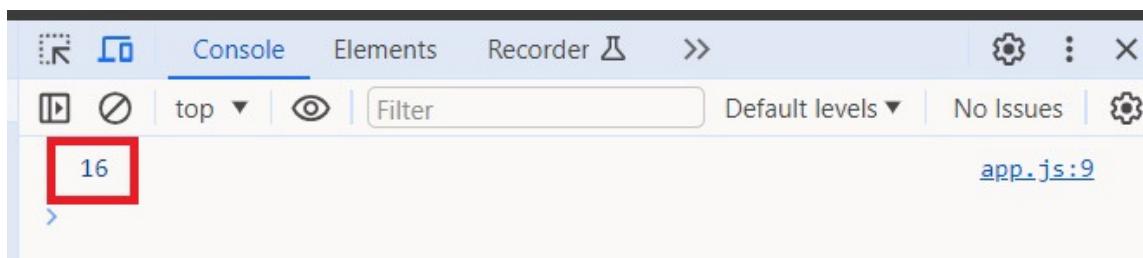


A screenshot of the Visual Studio Code interface. The top bar shows 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The title bar says 'app.js - Curso-FrontEnd'. The left sidebar has 'EXPLORER' and 'OPEN EDITORS 1 unsaved'. Under 'CURSO-FRONTEND', there are 'app.js' and 'index.html'. The main editor area contains the following code:

```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Como Acessar o(s) Caracter(es)
3
4 const houseDogName = 'Catorial'
5 const houseCatName = 'Bichano'
6
7 const houseAnimalsNames = houseDogName + ' ' + houseCatName
8
9 console.log(houseAnimalsNames.length)
10
```

The line '9 console.log(houseAnimalsNames.length)' is highlighted with a red rectangle.

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o número 16 é exibido.



/igor-rebolla

Strings

Comprimento de uma String

IMPORTANTÍSSIMO: O espaço em branco também conta como caractere.

Catiorro Bichano

8 1 7

Catiorro = 8 caractere

Bichano = 7 caractere

Espaço em branco = 1 caractere

Totalizando = 16 caractere conforme console.log da página anterior



Strings

Métodos de uma String

//MÉTODOS DE STRING

Antes de partirmos para os exemplos dos Métodos de uma string, é interessante conhecermos que existe uma pequena diferença técnica (porem importantíssimo saber) entre MÉTODOS E FUNÇÕES:

Função Uma função é basicamente um pedacinho de código que executa uma determinada ação específica.

Método: É uma função que está vinculada a um objeto ou tipo de dado em específico/particular.

Não se preocupem com isso nesse momento, falaremos mais sobre essas pequenas diferenças técnicas no decorrer das aulas.

Agora sim... podemos falar que uma string possui diversos métodos (várias funções que podem executar uma ação específica).

Por exemplo, tem um método de string chamado `toUpperCase` que vai fazer com que todos os caracteres da string fiquem em maiúsculos.

Então aqui abaixo vamos declarar um `console.log()` declarar a string no qual aplicaremos o método que é a `houseAnimalsNames` especificar um `.` e passaremos o nome do método que aqui é o `toUpperCase` e quando a vamos especificar uma propriedade como a `length` por exemplo a especificamos ponto `(.)` e o nome da propriedade, OK Pessoal!!!



/igor-rebolla

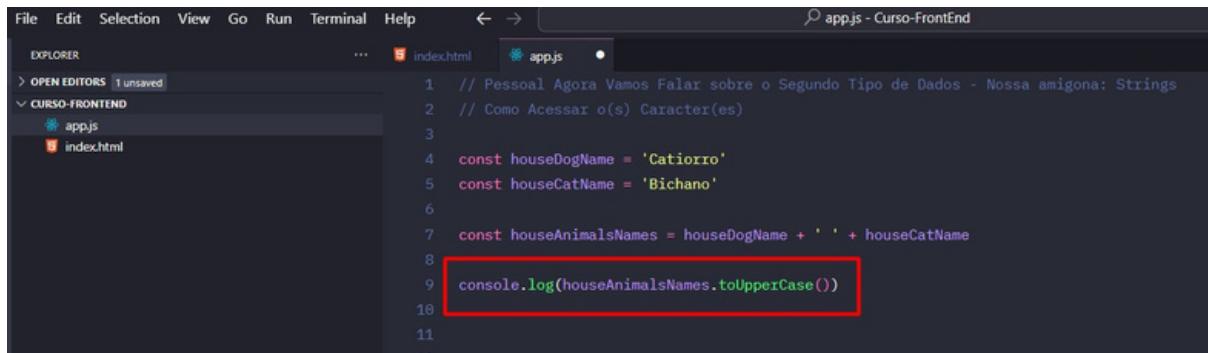
Strings

Métodos de uma String

//MÉTODOS DE STRING

Assim como `toUpperCase` é um método ou seja uma função que executa uma ação que faz com que todos os caracteres da string fiquem em maiúsculos, por ele ser um método nós temos que chamar esse método e fazemos isso especificando os parênteses no fim do nome do método. Então toda vez que você ver um ponto, um nome e esses parênteses no fim do nome significa que isso é um método de algo. Ficando assim:

```
console.log(houseAnimalsNames.toUpperCase())
```



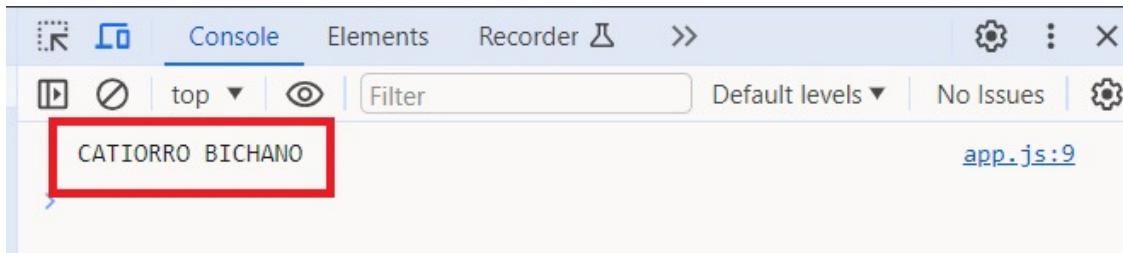
```
File Edit Selection View Go Run Terminal Help ← → ⌘ app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Como Acessar o(s) Caracter(es)
3
4 const houseDogName = 'Catiorro'
5 const houseCatName = 'Bichano'
6
7 const houseAnimalsNames = houseDogName + ' ' + houseCatName
8
9 console.log(houseAnimalsNames.toUpperCase())
10
11
12
```

Se a gente salvar esse arquivo, a gente vai ver no console que esse método pegou a string Catiorro Bichano e fez com que todos os caracteres dessa string ficassem em maiusculos.



Strings

Métodos de uma String



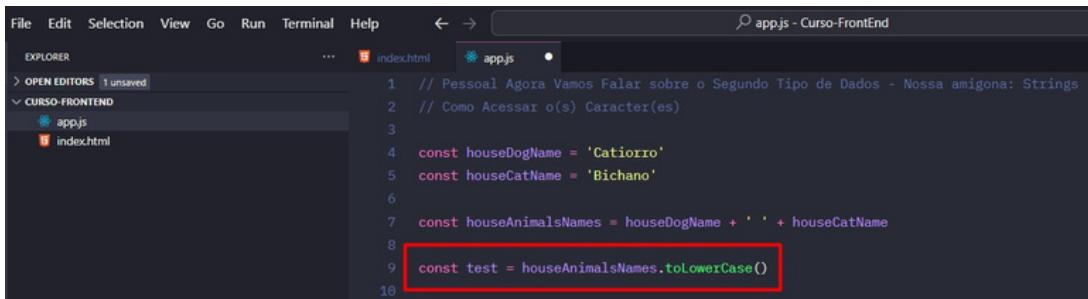
Tem um outro método que gostaria de mostrar para o conhecimento de vocês. Mas/porem/entretanto/todavia antes abaixo desse `console.log(houseAnimalsNames.toUpperCase())`, declaremos uma const test que receberá o resultado de `houseAnimalsNames.toLowerCase()` e como você deve ter pensado o método `toLowerCase()` fará com que todos os caracteres da string sejam minúsculos, só que ao invés de exibir o `houseAnimalsNames.toUpperCase()` no console, igual fizemos aqui na linha de cima, o que estamos fazendo é recebendo o valor que essa expressão `houseAnimalsNames.toLowerCase()` retorna e armazenando esse valor na const test. Então podemos utilizar em outro momento esse valor que a test armazena. Ficando assim:

```
const test = houseAnimalsNames.toLowerCase()
```



Strings

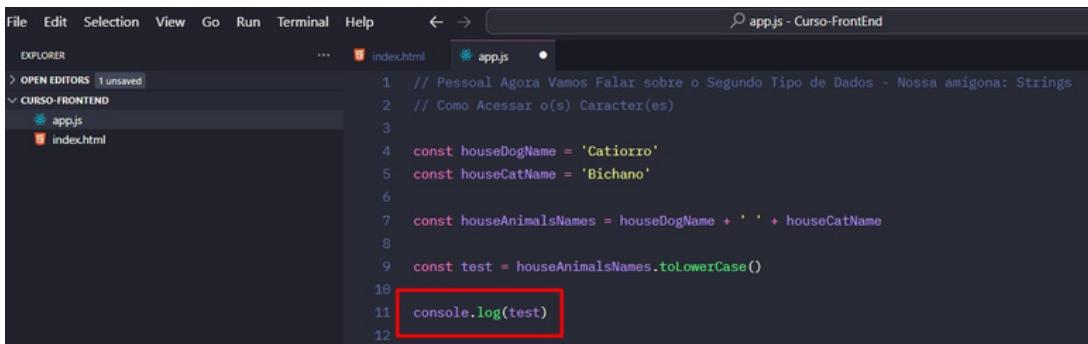
Métodos de uma String



```
File Edit Selection View Go Run Terminal Help ⌘ ⌘ index.html app.js
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Como Acessar o(s) Caracter(es)
3
4 const houseDogName = 'Catiorro'
5 const houseCatName = 'Bichano'
6
7 const houseAnimalsNames = houseDogName + ' ' + houseCatName
8
9 const test = houseAnimalsNames.toLowerCase()
```

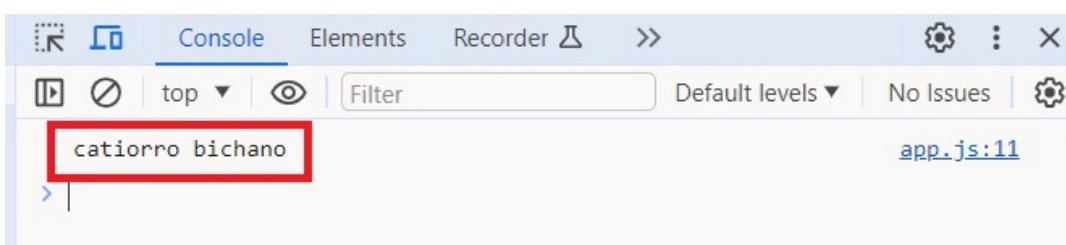
Vamos declarar um console.log que exibe a test. Ficando assim:

```
console.log(test)
```



```
File Edit Selection View Go Run Terminal Help ⌘ ⌘ index.html app.js
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Como Acessar o(s) Caracter(es)
3
4 const houseDogName = 'Catiorro'
5 const houseCatName = 'Bichano'
6
7 const houseAnimalsNames = houseDogName + ' ' + houseCatName
8
9 const test = houseAnimalsNames.toLowerCase()
10
11 console.log(test)
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto, podemos visualizar no console do Navegador que a string catiorro bichano está com todos os caracteres em minúsculos.



```
Console Elements Recorder > ⚙️ ⚙️ ×
top Filter Default levels No Issues
catorro bichano app.js:11
```



/igor-rebolla

Strings

Métodos de uma String

Gostaria de mostrar uma coisa para vocês, esses 2 métodos que a nós vimos, tanto o `toUpperCase` quanto o `toLowerCase` eles não alteram a string original(valor original).

Para ficar mais claro, exibiremos a `houseAnimalsNames` no console ao lado da test.

```
console.log(test, houseAnimalsNames)
```

E se salvarmos o arquivo, podemos visualizar que a `houseAnimalsNames` permanece intocável, ou seja a string continua do jeitinho que ela foi declarada com o C do nome em maiúsculo e o restante da palavra em minúsculo e com o B em maiúsculo e o restante da palavra em minúsculo.

Isso nos mostra que os métodos `toUpperCase` e o `toLowerCase` não mudam o seu valor original no qual eles estão executando uma ação.

Alguns métodos modificam, alguns métodos não modificam, OK!!



Strings

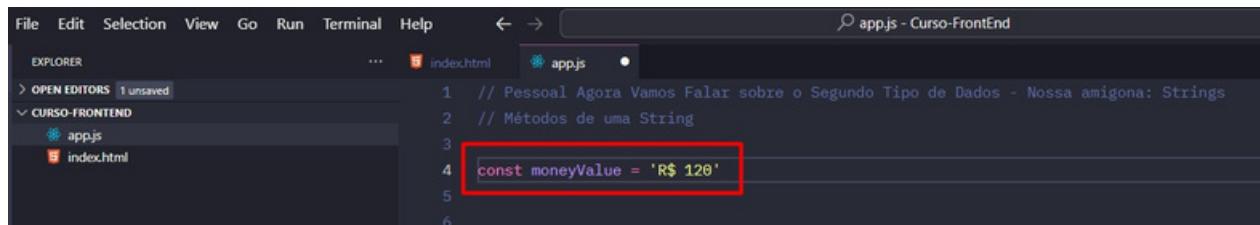
Métodos de uma String

Vamos ver agora o método indexOf.

E o que esse método indexOf vai fazer?

Ele vai buscar o index ou seja a posição do caractere que você determinar dentro dos parênteses, então vou dizer para o indexOf que eu gostaria que ele busque o caractere '\$' dentro da const moneyValue a qual criaremos aqui abaixo e presta bem atenção por gentileza que como a gente tá especificando que gostaríamos de uma buscar uma string, temos que passar esse \$ entre aspas.

```
const moneyValue = 'R$ 120'
```



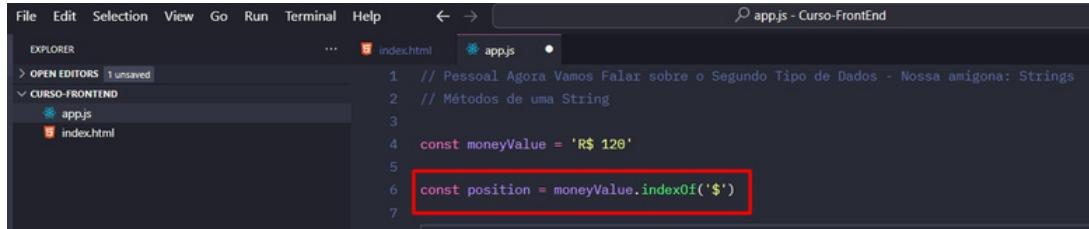
```
File Edit Selection View Go Run Terminal Help ← → / app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Métodos de uma String
3
4 const moneyValue = 'R$ 120'
5
6
```



Strings

Métodos de uma String

Esse carinha que estamos passando dentro dos parênteses, tem o nome de Argumento que aqui é representado pelo '\$'.



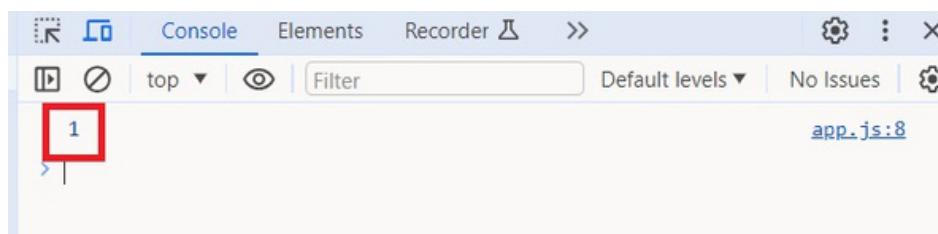
```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved
CURSO-FRONTEND app.js index.html
index.html app.js
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Métodos de uma String
3
4 const moneyValue = 'R$ 120'
5
6 const position = moneyValue.indexOf('$')
7
```

Vamos declarar um `console.log(position)`



```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved
CURSO-FRONTEND app.js index.html
index.html app.js
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Métodos de uma String
3
4 const moneyValue = 'R$ 120'
5
6 const position = moneyValue.indexOf('$')
7
8 console.log(position)
```

E ao salvarmos, a gente ve no console que o numero 1 será exibido.



```
Console Elements Recorder > Default levels ▾ No Issues
1
app.js:8
```



Strings

Métodos de uma String

Igor, Igor, Igor....E porque o 1 apareceu ali?

Porque se contarmos a posição de cada caractere dessa string moneyValue o \$ será o index 1.



/igor-rebolla

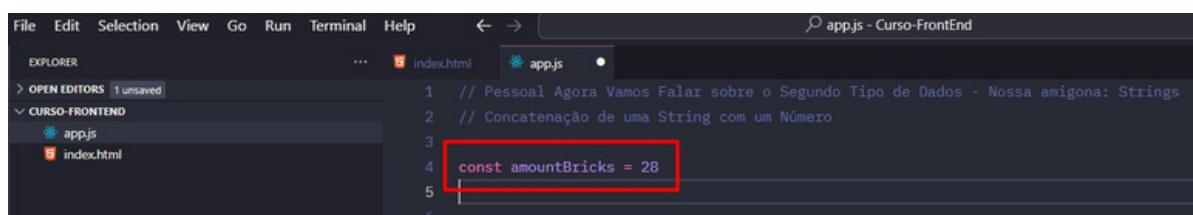
Strings

Concatenação de String com Número

Podemos concatenar string com o operador +, é possível fazer algo parecido também com o número se tentarmos adiciona-lo a string . Aqui abaixo vamos declarar uma const houseWall que recebe a string 'A casa possui ' + amountBricks + ' tijolos nas paredes.'

Quando concatenamos a string 'A casa possui ' com o número amountBricks nos bastidores(carinhosamente chamado de submundo) o JavaScript(JS) converte automaticamente esse número amountBricks em uma string, então o resultado de uma concatenação entre número e string, sempre vai ser string.

```
const amountBricks = 28
```



A screenshot of a code editor interface. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar says "app.js - Curso-FrontEnd". The left sidebar shows an "EXPLORER" section with "OPEN EDITORS 1 unsaved" and a "CURSO-FRONTEND" folder containing "app.js" and "index.html". The main code editor area has the following content:

```
// Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
// Concatenação de uma String com um Número
const amountBricks = 28
```

The line "const amountBricks = 28" is highlighted with a red rectangular box.

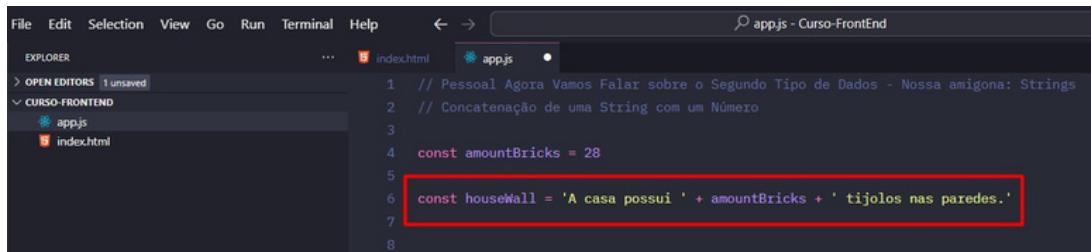


/igor-rebolla

Strings

Concatenação de String com Número

```
const houseWall = 'A casa possui ' + amountBricks + ' tijolos nas paredes.'
```



A screenshot of a code editor window titled "app.js - Curso-FrontEnd". The file contains the following code:

```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Concatenação de uma String com um Número
3
4 const amountBricks = 28
5
6 const houseWall = 'A casa possui ' + amountBricks + ' tijolos nas paredes.'
7
8
```

The line "const houseWall = 'A casa possui ' + amountBricks + ' tijolos nas paredes.'" is highlighted with a red box.

```
console.log(houseWall)
```



A screenshot of a code editor window titled "app.js - Curso-FrontEnd". The file contains the following code:

```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Concatenação de uma String com um Número
3
4 const amountBricks = 28
5
6 const houseWall = 'A casa possui ' + amountBricks + ' tijolos nas paredes.'
7
8 console.log(houseWall)
9
```

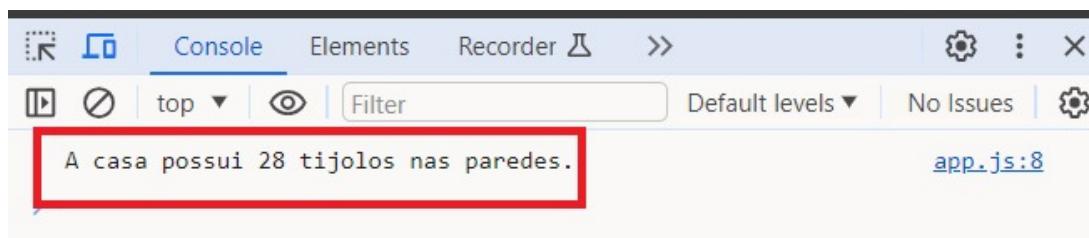
The line "console.log(houseWall)" is highlighted with a red box.

E ao salvarmos, a gente ve no console que a string concatenada com número A casa possui 28 tijolos nas paredes. é exibida.



Strings

Concatenação de String com Número



The screenshot shows the Chrome DevTools Console tab. The output window contains the following text:
A casa possui 28 tijolos nas paredes.
The line "A casa possui 28 tijolos nas paredes." is highlighted with a red rectangular box.



/igor-rebolla

Sugestões de prática:

1. Refazer e Entender os exemplos dessa aula;
2. Usar valores definidos por vocês para praticar mais sobre: Números e Strings

Colocar isso em prática no VSCode e ver o resultado.

Programação é prática, curiosidade e repetição!!!!



Concatenação de Strings (+1 exemplo)

Antes de partirmos para o aprendizado sobre Template Strings, veremos +1 exemplo sobre Concatenação de Strings. E faremos isso, por 2 motivos:

- 1º Para relembrarmos o conceito e praticarmos mais um pouquinho;
- 2º Para compararmos quando usarmos a Template Strings.

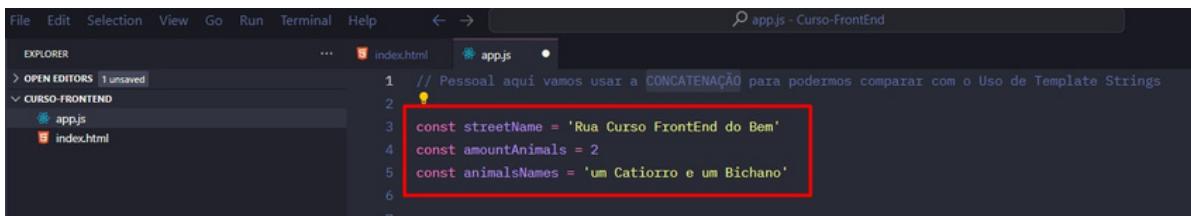
Bora abrir o nosso arquivo app.js e na primeira linha dentro desse arquivo criaremos um comentário: // Pessoal aqui vamos usar a CONCATENAÇÃO para podermos comparar com o Uso de Template Strings

E logo abaixo do comentário a segunda coisa que faremos é a declaração de 3 CONSTs. A primeira é uma const streetName que recebe uma string 'Rua Curso FrontEnd do Bem', daremos um enter para irmos até a linha abaixo e faremos a declaração da nossa segunda const, declararemos a const amountAnimals que recebe 2 e daremos um enter para irmos até a linha abaixo e faremos a declaração da nossa terceira const, declaremos a const animalsNames que recebe 'um Catiorro e um Bichano'. Ficando assim:

```
const streetName = 'Rua Curso FrontEnd do Bem'  
const amountAnimals = 2  
const animalsNames = 'um Catiorro e um Bichano'
```



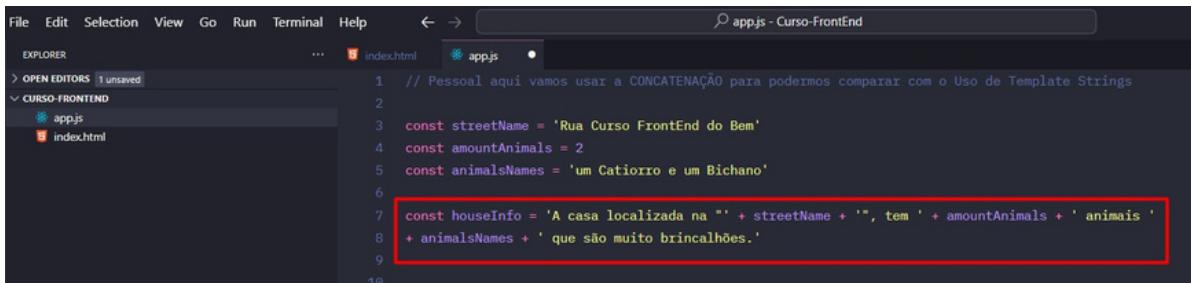
Concatenação de Strings (+1 exemplo)



```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
> OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal aqui vamos usar a CONCATENAÇÃO para podemos comparar com o Uso de Template Strings
2
3 const streetName = 'Rua Curso FrontEnd do Bem'
4 const amountAnimals = 2
5 const animalsNames = 'um Catiorro e um Bichano'
6
```

Criaremos uma String que contém essas 3 consts. Para fazermos isso, criaremos uma const chamada: houseInfo que vai receber 'A casa localizada na "' + streetName + '", tem ' + amountAnimals + ', animais ' + animalsNames + ' que são muito brincalhões.'. Ficando assim:

```
const houseInfo = 'A casa localizada na "' + streetName + '", tem ' + amountAnimals + ' animais ' + animalsNames + ' que são muito brincalhões.'
```



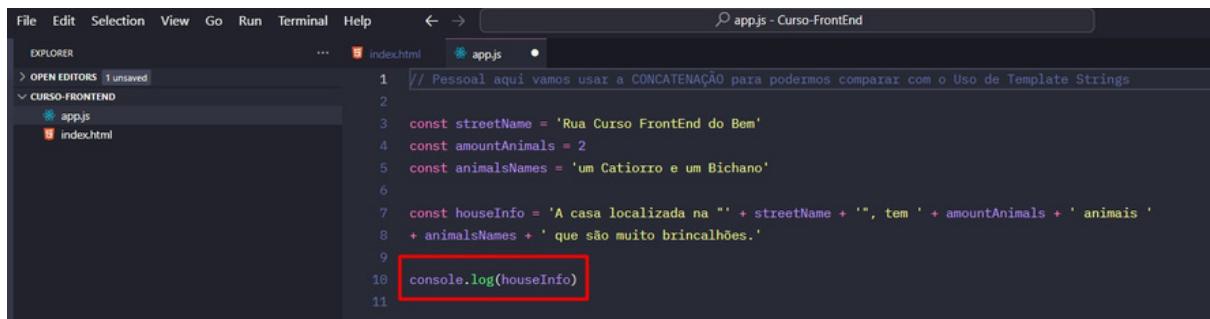
```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
> OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal aqui vamos usar a CONCATENAÇÃO para podemos comparar com o Uso de Template Strings
2
3 const streetName = 'Rua Curso FrontEnd do Bem'
4 const amountAnimals = 2
5 const animalsNames = 'um Catiorro e um Bichano'
6
7 const houseInfo = 'A casa localizada na "' + streetName + '", tem ' + amountAnimals + ' animais '
8 + animalsNames + ' que são muito brincalhões.'
9
```

Adicionaremos um console.log passando a houseInfo. Ficando assim:

```
console.log(houseInfo)
```

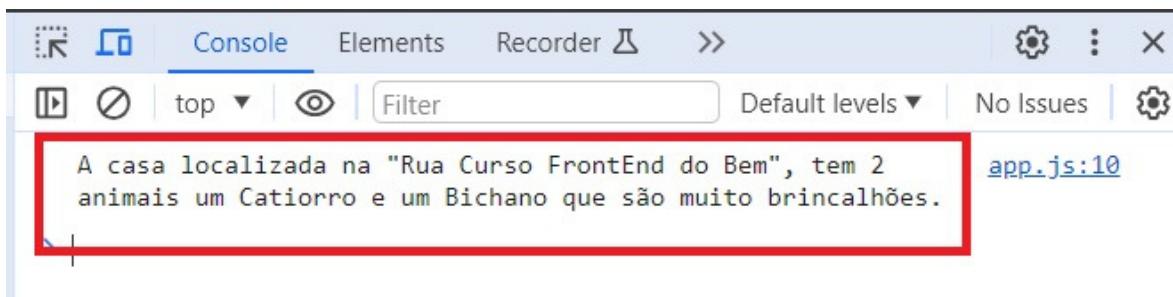


Concatenação de Strings (+1 exemplo)



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal aqui vamos usar a CONCATENAÇÃO para podermos comparar com o Uso de Template Strings
2
3 const streetName = 'Rua Curso FrontEnd do Bem'
4 const amountAnimals = 2
5 const animalsNames = 'um Catiorro e um Bichano'
6
7 const houseInfo = 'A casa localizada na "' + streetName + '", tem ' + amountAnimals + ' animais '
8 + animalsNames + ' que são muito brincalhões.'
9
10 console.log(houseInfo)
11
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que a frase é exibida.



```
A casa localizada na "Rua Curso FrontEnd do Bem", tem 2 animais um Catiorro e um Bichano que são muito brincalhões.
app.js:10
```

No exemplo acima podemos ver a quantidade de abre e fecha aspas que nós utilizamos dentro de um exemplo considerado “simples” e de concatenações que a string do exemplo (USANDO CONCATENAÇÃO) tem, e com base nisso verificamos que essa string não é muito amigável se quisermos dar alguma manutenção nela e até durante o seu desenvolvimento.



Template Strings

Veremos uma maneira “diferentona” para escrevermos STRINGS, a qual chamamos de: TEMPLATE STRINGS

Pessoal essa aqui é a TEMPLATE STRINGS, TEMPLATE STRINGS esse aqui é o Pessoal.

Uma das coisas mais “camaradas” que uma Template String possibilita é a de inserirmos as nossas amigas variáveis dentro da string sem ter que fechar uma string, declarar um +, abrir outra string, declarar outro +, fechar string e etc... Parece até o Daniel San em Karatê Kid (põe casaco, tira casaco, põe casaco....).

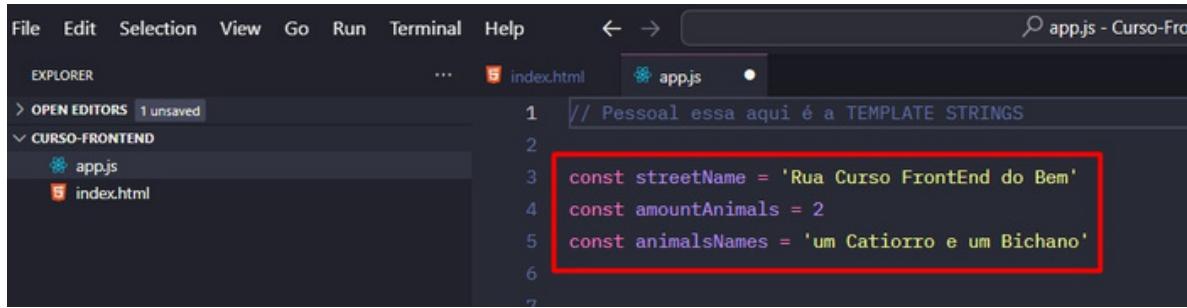
Então Boraaaaaa abrir o nosso arquivo app.js e na primeira linha dentro desse arquivo criaremos um comentário: // Pessoal essa aqui é a TEMPLATE STRINGS

E logo abaixo do comentário a segunda coisa que faremos é a declaração de 3 CONSTS. A primeira é uma const streetName que recebe 'Rua Curso FrontEnd do Bem', daremos um enter para irmos até a linha abaixo e faremos a declaração da nossa segunda const, declararemos a const amountAnimals que recebe 2 e daremos um enter para irmos até a linha abaixo e faremos a declaração da nossa terceira const, declararemos a const AnimalsNames que recebe 'um Catiorro e um Bichano'. Ficando assim:

```
const streetName = 'Rua Curso FrontEnd do Bem'  
const amountAnimals = 2  
const animalsNames = 'um Catiorro e um Bichano'
```



Template Strings



A screenshot of the Visual Studio Code interface. The title bar says "app.js - Curso-FrontEnd". The left sidebar shows an "EXPLORER" view with "OPEN EDITORS 1 unsaved" and a folder "CURSO-FRONTEND" containing "app.js" and "index.html". The main editor area has the following code:

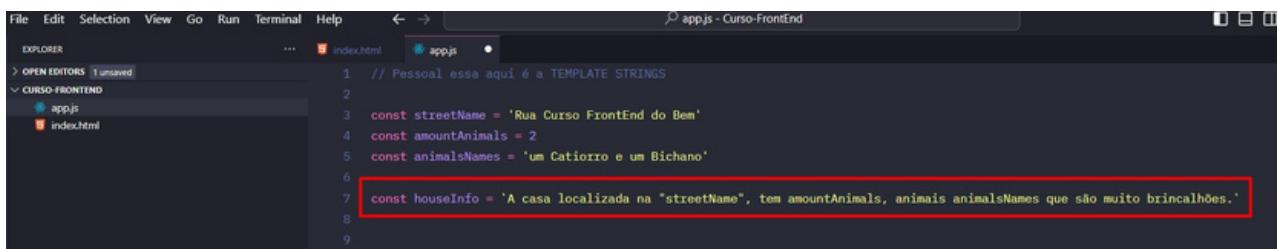
```
// Pessoal essa aqui é a TEMPLATE STRINGS
const streetName = 'Rua Curso FrontEnd do Bem'
const amountAnimals = 2
const animalsNames = 'um Catiorro e um Bichano'
```

The last three lines of code are highlighted with a red box.

Declaremos uma const chamada: houseInfo que receberá: abertura e fechamento de crases " e dentro dessas crases escreveremos a mesma mensagem utilizada em Concatenação de Strings (conforme vimos na página 3 dessa aula) `A casa localizada na "streetName", tem amountAnimals, animais animalsNames que são muito brincalhões.` Ficando assim:

```
const houseInfo = `A casa localizada na "streetName", tem amountAnimals, animais animalsNames que são muito brincalhões. `
```

IMPORTANTÍSSIMO: Na streetName, na amountAnimals e na animalsNames o que desejamos é que essas 3 variáveis sejam inseridas aqui. Se deixarmos desse jeitinho essa const houseInfo = `A casa localizada na "streetName", tem amountAnimals, animais animalsNames que são muito brincalhões. ` é uma string considerada normal que cumprirá a mesma função de uma string que possui abertura e fechamento de aspas E não é isso que queremos, certo!!!!



A screenshot of the Visual Studio Code interface. The title bar says "app.js - Curso-FrontEnd". The left sidebar shows an "EXPLORER" view with "OPEN EDITORS 1 unsaved" and a folder "CURSO-FRONTEND" containing "app.js" and "index.html". The main editor area has the following code:

```
// Pessoal essa aqui é a TEMPLATE STRINGS
const streetName = 'Rua Curso FrontEnd do Bem'
const amountAnimals = 2
const animalsNames = 'um Catiorro e um Bichano'
const houseInfo = `A casa localizada na "streetName", tem amountAnimals, animais animalsNames que são muito brincalhões. `
```

The last line of code is highlighted with a red box.

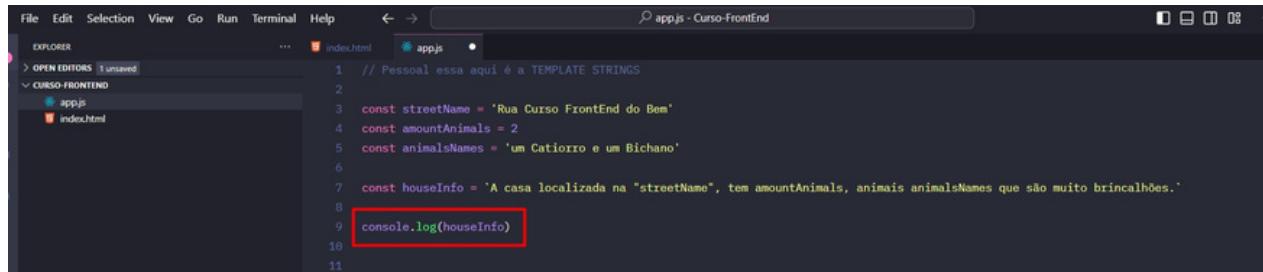


/igor-rebolla

Template Strings

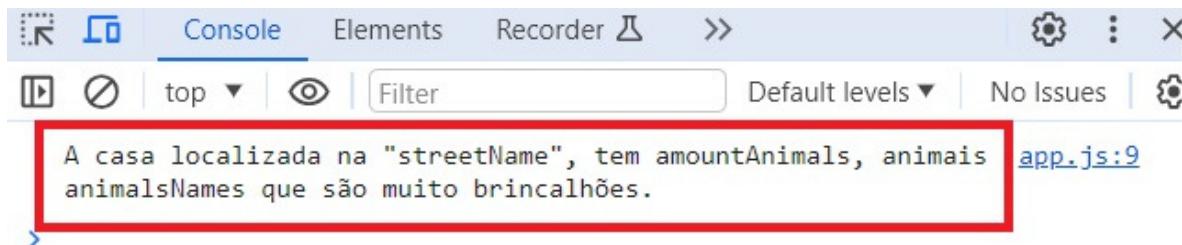
Adicionaremos um console.log passando a houseInfo. Ficando assim:

```
console.log(houseInfo)
```



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
OPEN EDITORS 3 unsaved
EXPLORER
CORSO-FRONTEND
  app.js
  index.html
1 // Pessoal essa aqui é a TEMPLATE STRINGS
2
3 const streetName = 'Rua Curso FrontEnd do Bem'
4 const amountAnimals = 2
5 const animalsNames = 'um Catiorro e um Bichano'
6
7 const houseInfo = `A casa localizada na "streetName", tem amountAnimals, animais ${animalsNames} que são muito brincalhões.`
8
9 console.log(houseInfo)
10
11
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que streetName, amountAnimals e animalsNames foram exibidas como strings, só que com palavras(literalmente) e não inseridas como gostaríamos que fossem.



```
Console Elements Recorder > Default levels ▾ No Issues
▶ top ▾ Filter
A casa localizada na "streetName", tem amountAnimals, animais app.js:9 animalsNames que são muito brincalhões.
```



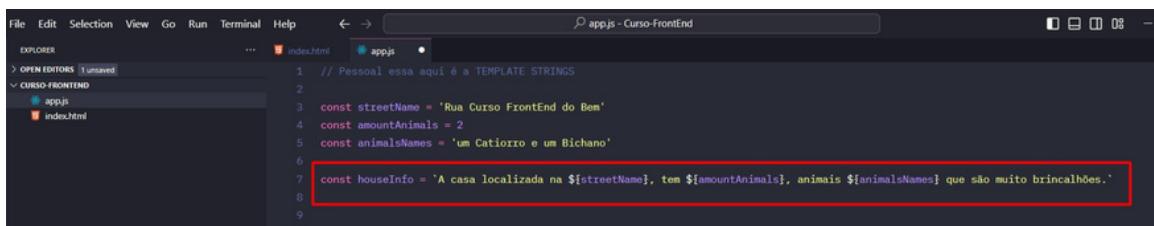
/igor-rebolla

Template Strings

Igor, Igor... então como vamos fazer para que essas variáveis sejam inseridas da forma correta?

Ótima pergunta: Dentro de uma Template Strings pra inserirmos uma variável, precisamos envolver-la com um \$ e {}, pegaremos a streetName, declararemos um \$ abriremos e fecharemos chaves {} e inserimos a streetName dentro dessas chaves "\${streetName}", faremos a mesma coisa com a amountAnimals, declararemos um \$ abriremos e fecharemos chaves {} e inserimos a amountAnimals dentro dessas chaves "\${amountAnimals}", faremos igualzinho com a animalsNames, declararemos um \$ abriremos e fecharemos chaves e inserimos a animalsNames dentro dessas chaves "\${animalsNames}". Ficando assim:

```
const houseInfo = `A casa localizada na ${streetName}, tem  
${amountAnimal}, animais ${animalsNames} que são muito brincalhões.`
```



A screenshot of a code editor (VS Code) showing a file named 'app.js'. The code contains a template string assignment:

```
// Pessoal essa aqui é a TEMPLATE STRINGS
const streetName = 'Rua Curso FrontEnd do Bem'
const amountAnimals = 2
const animalsNames = 'um Catiorro e um Bichano'
const houseInfo = `A casa localizada na ${streetName}, tem ${amountAnimals}, animais ${animalsNames} que são muito brincalhões.`

```

The last line of code, which defines the template string, is highlighted with a red rectangle.

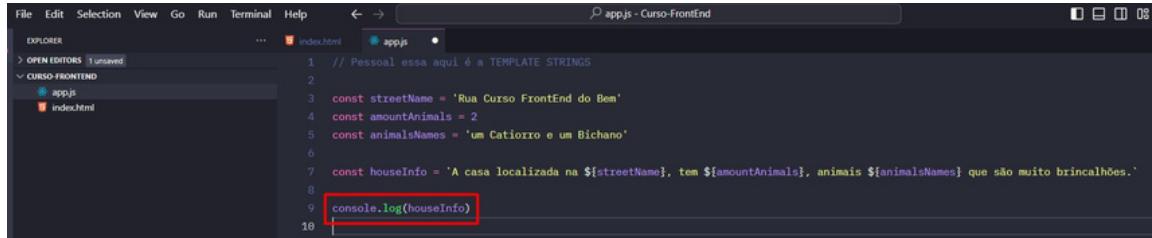
E dando aquela olhada marota na linha acima podemos verificar que existe uma boa diferença de sintaxe entre uma TEMPLATE STRINGS (facilitando d+++++) a nossa leitura e uma STRING NORMAL com abertura e fechamento de aspas.



Template Strings

Adicionaremos um `console.log` passando a `houseInfo`. Ficando assim:

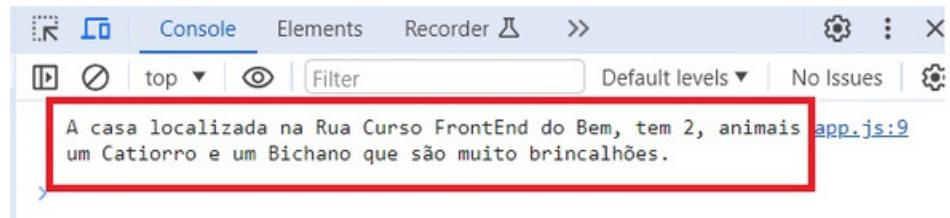
```
console.log(houseInfo)
```



A screenshot of a code editor showing an `app.js` file. The code contains template strings and a `console.log` statement. A red box highlights the `console.log` line.

```
// Pessoal essa aqui é a TEMPLATE STRINGS
const streetName = 'Rua Curso FrontEnd do Bem'
const amountAnimals = 2
const animalsNames = 'um Catiorro e um Bichano'
const houseInfo = `A casa localizada na ${streetName}, tem ${amountAnimals}, animais ${animalsNames} que são muito brincalhões.`
console.log(houseInfo)
```

E ao salvarmos o arquivo `app.js` com aquele `CTRL + S` maroto podemos ver no `console` do Navegador(Chrome) que os valores das 3 `consts` foram inseridos corretamente dentro da nossa string.



A screenshot of the Chrome DevTools Console tab. It shows the output of the `console.log` statement from the previous screenshot. The output is a template string: "A casa localizada na Rua Curso FrontEnd do Bem, tem 2, animais [app.js:9](#) um Catiorro e um Bichano que são muito brincalhões.". This output is highlighted with a red box.



/igor-rebolla

Template HTML

E aqui que vem o “jump of the cat”(pulo do gato).... Podemos usar também a TEMPLATE STRINGS para isso.

Bora criar mais um comentário:// Pessoal esse aqui é o brother
TEMPLATE(S) HTML

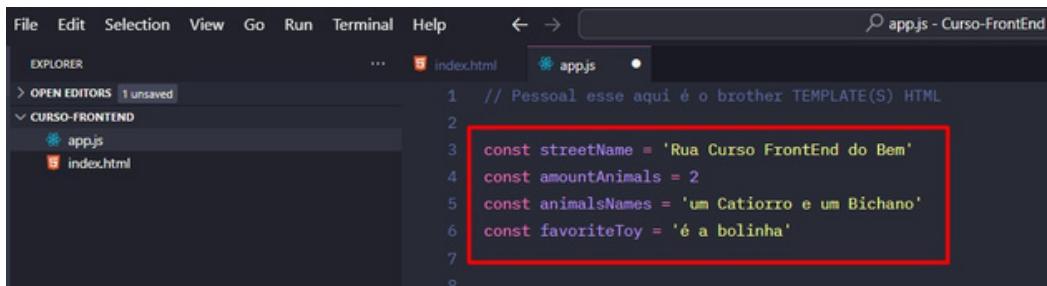
Então abaixo do comentário // Pessoal esse aqui é o brother TEMPLATE(S) HTML . Usaremos as 3 CONSTs dos exemplos anteriores, só que aqui acrescentaremos +1 const, a qual chamaremos de favoriteToy que receberá 'é a bolinha' .

```
const streetName = 'Rua Curso FrontEnd do Bem'  
const amountAnimals = 2  
const animalsNames = 'um Catiorro e um Bichano'  
const favoriteToy = 'é a bolinha'
```

Ficando assim:



Template HTML



The screenshot shows a code editor interface with a dark theme. At the top, there's a menu bar with File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. Below the menu is an Explorer sidebar with 'OPEN EDITORS' (1 unsaved) and a tree view showing 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main area contains a code editor with the following content:

```
// Pessoal esse aqui é o brother TEMPLATE(S) HTML
const streetName = 'Rua Curso FrontEnd do Bem'
const amountAnimals = 2
const animalsNames = 'um Catiorro e um Bichano'
const favoriteToy = 'é a bolinha'
```

A red box highlights the template string assignment on line 3.

Criaremos uma const chamada: houseInfoTemplate que receberá uma template string, abriremos e fecharemos crase `` , ah antes....simbora dar um enter maroto aqui para quebrarmos uma linha dentro da abertura e fechamento das crases, para deixarmos nossas informações visualmente mais legíveis.

```
const houseInfoTemplate = `
```

Criaremos um <h3> e fecharemos esse </h3> e dentro desse h3 teremos a "\${streetName}" e na linha abaixo desse h3, criaremos o nosso primeiro <p> e fecharemos esse </p> e dentro desse p escreveremos: tem e a \${amountAnimals}" e na linha abaixo desse p, declaremos um segundo <p> e fecharemos esse segundo </p> e dentro dele escreveremos: animais "\${animalsNames}" que são muito brincalhões e na linha abaixo desse segundo p, declararemos um spam <spam> e fecharemos esse e dentro dele escreveremos: e o nosso brinquedo favorito \${favoriteToy}

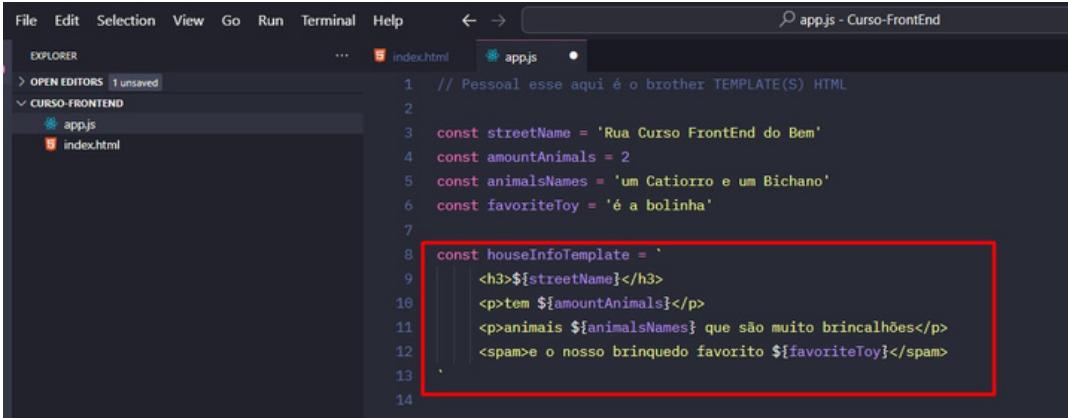
Ficando assim:

```
const houseInfoTemplate = `
```

```
<h3>${streetName}</h3>
<p>tem ${amountAnimals}</p>
<p>animais ${animalsNames} que são muito brincalhões</p>
<spam>e o nosso brinquedo favorito ${favoriteToy}</spam>
```



Template HTML

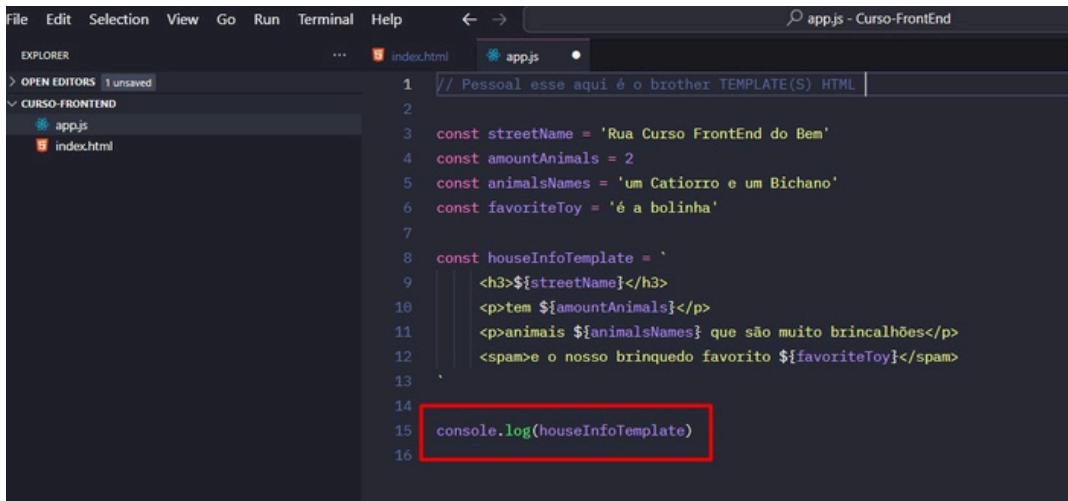


```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui é o brother TEMPLATE(S) HTML
2
3 const streetName = 'Rua Curso FrontEnd do Bem'
4 const amountAnimals = 2
5 const animalsNames = 'um Catiorro e um Bichano'
6 const favoriteToy = 'é a bolinha'
7
8 const houseInfoTemplate =
9   <h3>${streetName}</h3>
10  <p>tem ${amountAnimals}</p>
11  <p>animais ${animalsNames} que são muito brincalhões</p>
12  <spam>e o nosso brinquedo favorito ${favoriteToy}</spam>
13
14
15
16
```

The code editor shows a file named 'app.js' with a template literal. The template literal contains several interpolation statements (`\${}`) that will be replaced by variables defined above it. A red box highlights the template literal from line 8 to line 13.

Adicionaremos um `console.log` passando a `houseInfoTemplate`. Ficando assim:

```
console.log(houseInfoTemplate)
```



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui é o brother TEMPLATE(S) HTML
2
3 const streetName = 'Rua Curso FrontEnd do Bem'
4 const amountAnimals = 2
5 const animalsNames = 'um Catiorro e um Bichano'
6 const favoriteToy = 'é a bolinha'
7
8 const houseInfoTemplate =
9   <h3>${streetName}</h3>
10  <p>tem ${amountAnimals}</p>
11  <p>animais ${animalsNames} que são muito brincalhões</p>
12  <spam>e o nosso brinquedo favorito ${favoriteToy}</spam>
13
14
15 console.log(houseInfoTemplate)
16
```

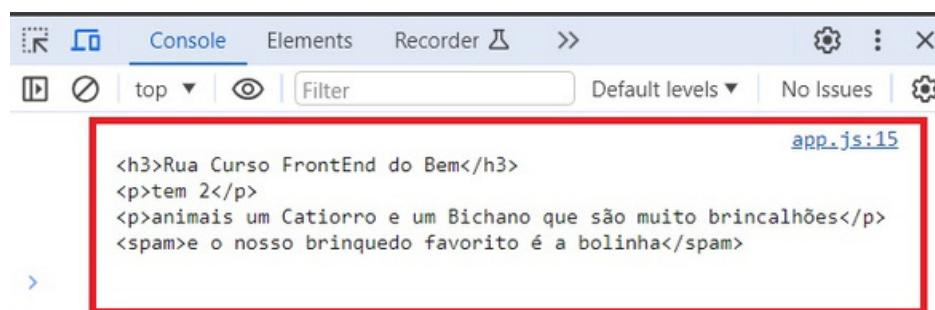
The code editor shows the same 'app.js' file with an additional line at the bottom: `console.log(houseInfoTemplate)`. A red box highlights this new line.



/igor-rebolla

Template HTML

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver o nosso Template HTML(houseInfoTemplate) sendo exibido no console do Navegador(Chrome)



```
app.js:15
<h3>Rua Curso FrontEnd do Bem</h3>
<p>tem 2</p>
<p>animais um Catiorro e um Bichano que são muito brincalhões</p>
<spam>e o nosso brinquedo favorito é a bolinha</spam>
```

Igor, Igor, Igor, Igor..... surgiu uma dúvida aqui: quando eu uso Template Strings e quando eu uso as strings.... igual o exemplo da concatenação?

Essa pergunta é muito boa, o Igor aqui fica feliz com esse tipo de questionamento: Normalmente quando não precisamos inserir variáveis na string ou fazermos uma quebra de linha não tem muito sentido usar Template Strings. E se durante a criação da string percebermos que ela necessita ter variáveis ou necessita ter aspas ou é um Template Html, a Template Strings vai facilitar e muitoooo tanto a manutenção quanto o código que você estiver escrevendo.

Só reforçando, tanto as Strings (aspas simples e duplas) quanto as Template Strings e não podemos deixar de lado o Template HTML, nós ainda falaremos desses 3 carinhas no decorrer das nossas aulas. Fiquem tranquilos quanto a isso!!!!



Template HTML

Uma última observação que eu gostaria de passar aqui nessa aula é a seguinte: Vocês notaram que quando falamos de Template HTML, além das constantes e das crases, também voltaram a aparecer um `<h3>`, dois parágrafos `<p>` e um `<spam>`

Lembram que vimos isso nas aulas de HTML (o primeiro pilar desse curso) e que agora será que elas “surgiram do nada” dentro do JavaScript (o terceiro pilar desse curso).

A resposta é Nãooooooooooooooo, os 3 pilares desse curso estão interligados, por isso eu “bato nessa tecla”, aprender a base da base é importantíssimo (que é o intuito desse curso), pois assim quando os pilares começarem a se juntar, tudo vai ficar mais claro para vocês, mesmo quando as coisas começarem a ficar mais complexas.... ah eu lembro que na aula 5 do Curso FrontEnd do Bem eu aprendi que o conceito é esse e o exemplo é esse, então se eu juntar isso com o conteúdo da aula 20 por exemplo, o resultado será esse que estamos vendo aqui (Template HTML).

É exatamente assim que as coisas começam a se encaixar dentro do mundo da programação!!!



Sugestões de prática:

1. Refazer e Entender os exemplos dessa aula;
2. Usar valores definidos por vocês para praticar mais sobre: Template Strings e Template HTML;
3. Acrescentar 1 nova const da sua preferência em Template Strings e remover 2 consts em Template HTML.

Colocar isso em prática no VSCode e ver o resultado no console do Navegador.

Programação é prática, curiosidade e repetição!!!!



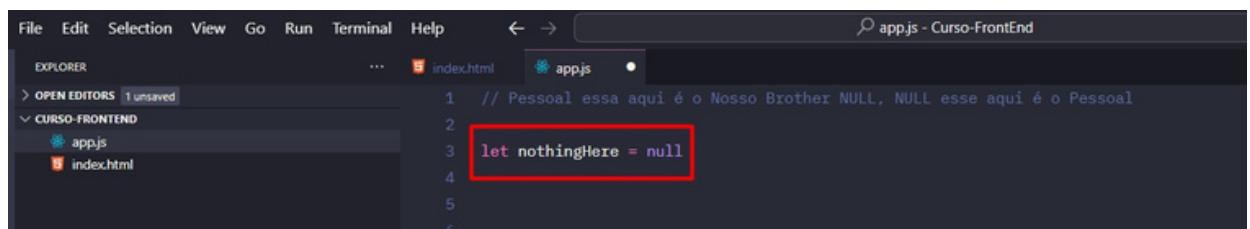
Null

Igor, Igor, Igor..... Quando que esse tal de NULL será utilizado?
Aeeee gostei, logo no inicio da aula uma ótima pergunta:
Quando quisermos indicar de forma intencional que não tem nenhum valor
em uma variável.

Ah, antes disso..... dentro do nosso ilustríssimo arquivo app.js, na linha 1
criaremos um comentário // Pessoal esse aqui é o Nosso Brother NULL,
NULL esse aqui é o Pessoal.

Então logo abaixo do comentário // Pessoal esse aqui é o Nosso Brother
NULL, NULL esse aqui é o Pessoal, declararemos uma let chamada
nothingHere e atribuiremos null pra ela. Ficando assim:

```
let nothingHere = null
```



```
File Edit Selection View Go Run Terminal Help ← → ⌘ app.js - Curso-FrontEnd
EXPLORER
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
```

```
1 // Pessoal essa aqui é o Nosso Brother NULL, NULL esse aqui é o Pessoal
2
3 let nothingHere = null
4
5
6
```

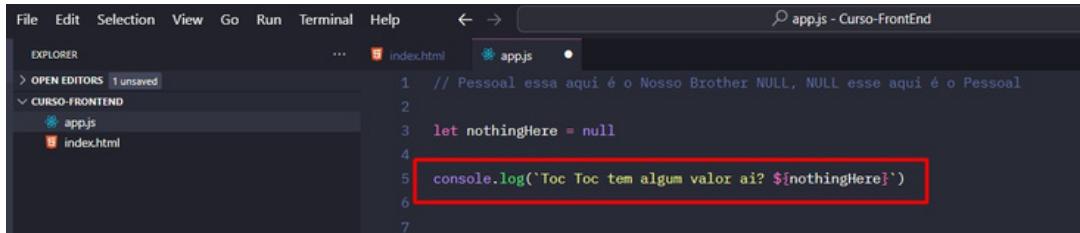
Agora para um melhor entendimento, faremos 3 exemplos distintos:

E o primeiro deles será uma template strings (já que estamos com template strings “fresquinha” na cabeça, a qual vimos na aula 20 - anterior do curso.). Declararemos uma template string “Toc Toc tem algum valor ai? e o valor da variável nothingHere \${nothingHere} dentro de um console.log().
Ficando assim:

```
console.log(`Toc Toc tem algum valor ai? ${nothingHere}`)
```



Null



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal essa aqui é o Nosso Brother NULL, NULL esse aqui é o Pessoal
2
3 let nothingHere = null
4
5 console.log(`Toc Toc tem algum valor ai? ${nothingHere}`)
6
7
```

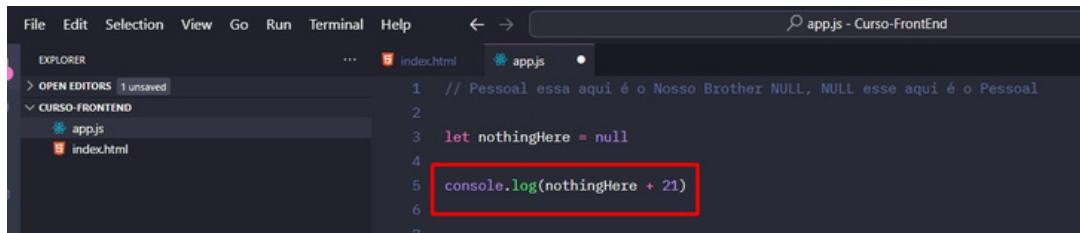
E ao salvarmos o arquivo app.js com aquele CTRL + S maroto, podemos ver no console do Navegador(Chrome) que NULL foi convertido para STRING.



```
Console Elements Recorder > ⚙️ ⚙️
▶ top Filter Default levels No Issues
Toc Toc tem algum valor ai? null app.js:5
>
```

E o segundo exemplo que veremos é a variável nothingHere + o número 21 dentro de um console.log(). Ficando assim:

```
console.log(nothingHere + 21)
```



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal essa aqui é o Nosso Brother NULL, NULL esse aqui é o Pessoal
2
3 let nothingHere = null
4
5 console.log(nothingHere + 21)
6
7
```



/igor-rebolla

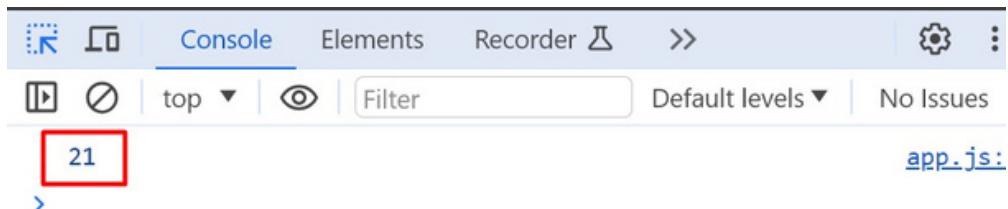
Null

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto, podemos ver que no console do Navegador(Chrome) apareceu um número 21 que parece meio perdidão.

Eita Igor, e porque esse 21 apareceu aqui....certeza que ele não tá perdidão?)

Hoje vocês estão bons de perguntar eihh... estou gostando de ver:

Quando alguma operação matemática que envolve NULL é executada, ele de forma automática é interpretado como 0.



E o terceiro exemplo que veremos é somente a variável nothingHere dentro de um console.log(). Ficando assim:

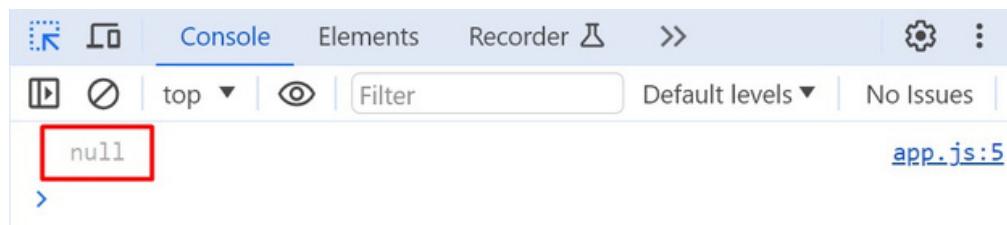
```
console.log(nothingHere)
```

A screenshot of the Visual Studio Code (VS Code) interface. The file 'app.js' is open in the editor. The code contains a single line: 'console.log(nothingHere)'. This line is highlighted with a red box.

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que null será exibido



Null



Undefined

Igor, Igor Igor... Acho que sabemos quando o UNDEFINED será utilizado, tomando como base a explicação do Null. Vê se estamos certos Beleza?
Ok, o Igor tá prestando atenção, bora lá....

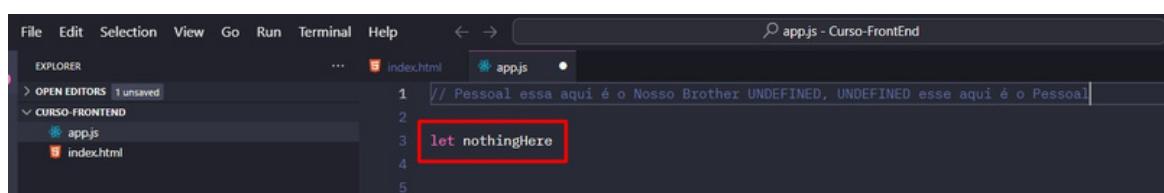
Oh é assim se com o Null o valor é atribuído de forma intencional, então quando é UNDEFINED, o nosso terceiro pilar do curso que aqui no caso é o JavaScript(JS), ele atribui de forma automática esse valor em uma variável que ainda não tem nenhum valor guardado nela

Eita que o Igor tá orgulhoso demais de vocês, isso mesmo todos vocês que estão lendo esse material... sinal que estamos no caminho certo..... Então agora que vocês deram aula.... bora ver esse tal de Undefined na prática.

Ah, só que como de costume (para treinarmos comentários), antes dentro do nosso ilustríssimo arquivo app.js, na linha 1 vamos criar um comentário // Pessoal esse aqui é o Nosso Brother UNDEFINED, UNDEFINED esse aqui é o Pessoal

Então logo abaixo do comentário // Pessoal esse aqui é o Nosso Brother UNDEFINED, UNDEFINED esse aqui é o Pessoal, declararemos uma let chamada nothingHere, só que não atribuiremos nenhum valor para ela. Ficando assim:

```
let nothingHere
```



```
// Pessoal essa aqui é o Nosso Brother UNDEFINED, UNDEFINED esse aqui é o Pessoal
let nothingHere
```



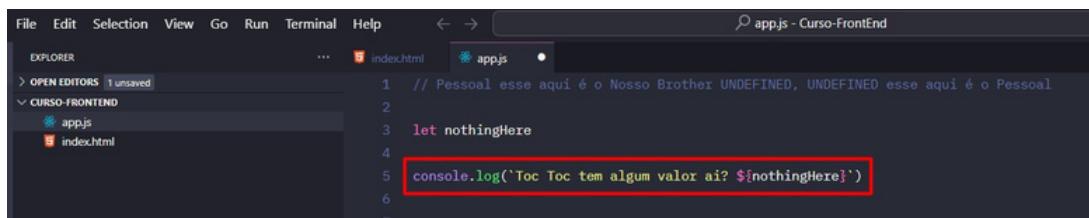
/igor-rebolla

Undefined

Assim como fizemos com NULL.... aqui em UNDEFINED, também usaremos 3 exemplos para um melhor entendimento.

E o primeiro deles será uma template strings (já que estamos com a template strings “fresquinha” na cabeça, a qual vimos na aula 20 e também em Null dentro dessa aula 21), bora declarar uma template string “Toc Toc tem algum valor ai? e o valor da variável nothingHere \${nothingHere} dentro de um console.log(). Ficando assim:

```
console.log(`Toc Toc tem algum valor ai? ${nothingHere}`)
```

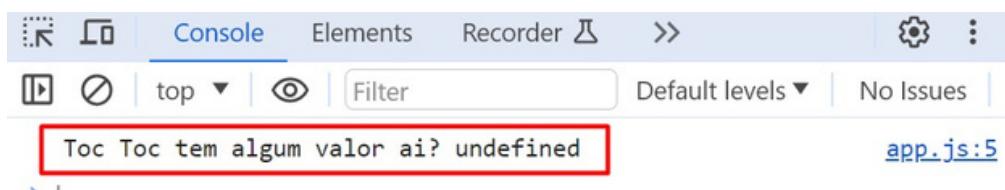


A screenshot of the Visual Studio Code interface. The title bar says "app.js - Curso-FrontEnd". The left sidebar shows an "EXPLORER" view with "OPEN EDITORS 1 unsaved" and a tree structure for "CURSO-FRONTEND" containing "app.js" and "index.html". The main editor area has the following code:

```
1 // Pessoal esse aqui é o Nosso Brother UNDEFINED, UNDEFINED esse aqui é o Pessoal
2
3 let nothingHere
4
5 console.log(`Toc Toc tem algum valor ai? ${nothingHere}`)
6
```

The line `console.log(`Toc Toc tem algum valor ai? \${nothingHere}`)` is highlighted with a red box.

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que ao inserirmos undefined como uma variável dentro de uma string que undefined é convertido para uma string, então esse é o nosso brother undefined um valor que o JavaScript atribui de forma automática para uma variável que ainda não guarda nenhum valor.



A screenshot of the Chrome DevTools Console tab. The tabs at the top are "Console" (which is selected), "Elements", and "Recorder". Below the tabs are buttons for "Play", "Stop", "top", "Filter", "Default levels", and "No Issues". The main area shows the output of the previous template string code:

```
Toc Toc tem algum valor ai? undefined
```

The output is highlighted with a red box. To the right of the output, it says "app.js:5".

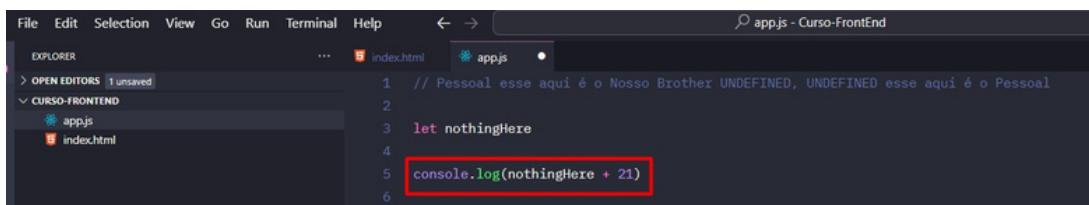


/igor-rebolla

Undefined

E o segundo exemplo que veremos é a variável nothingHere + o número 21 dentro de um console.log(). Ficando assim:

```
console.log(nothingHere + 21)
```

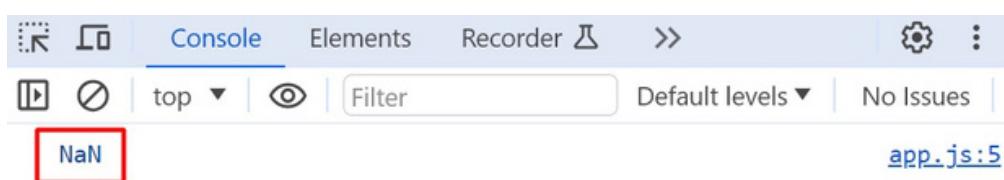


A screenshot of the Visual Studio Code interface. The title bar says "app.js - Curso-FrontEnd". The left sidebar shows an "EXPLORER" view with "OPEN EDITORS 1 unsaved" and a tree structure for "CURSO-FRONTEND" containing "app.js" and "index.html". The main editor area has the following code:

```
// Pessoal esse aqui é o Nosso Brother UNDEFINED, UNDEFINED esse aqui é o Pessoal
let nothingHere
console.log(nothingHere + 21)
```

The line "console.log(nothingHere + 21)" is highlighted with a red rectangle.

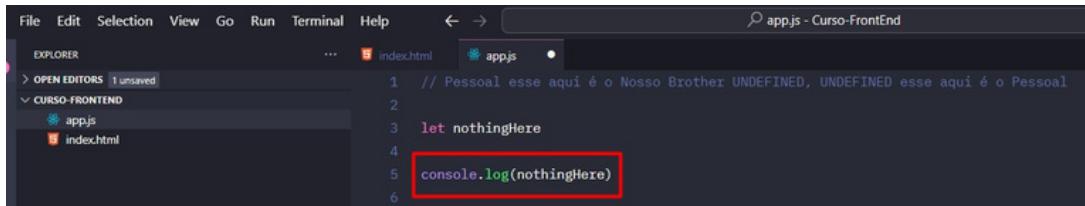
E ao salvarmos o arquivo app.js com aquele CTRL + S maroto, podemos ver no console do Navegador(Chrome) que ao tentarmos utilizar o undefined (já que a nothingHere é undefined) em uma expressão numérica que no nosso exemplo aqui é: nothingHere + 21, o resultado obtido será um: Nan (Not a Number)



Undefined

E o terceiro exemplo que veremos é somente a variável `nothingHere`. Ficando assim:

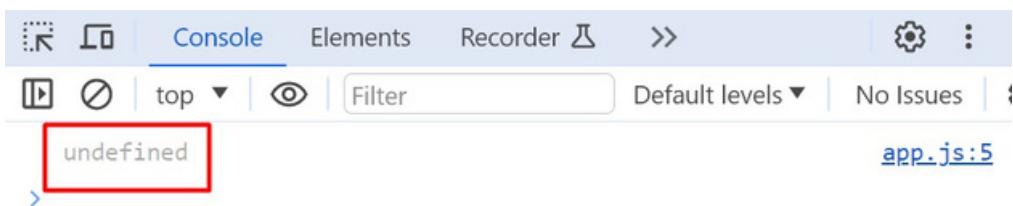
```
console.log(nothingHere)
```



```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved
CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui é o Nosso Brother UNDEFINED, UNDEFINED esse aqui é o Pessoal
2
3 let nothingHere
4
5 console.log(nothingHere)
6
```

E ao salvarmos o arquivo `app.js` com aquele `CTRL + S` maroto podemos ver no console do Navegador(Chrome) que `nothingHere` é `undefined`. E porque isso acontece?

Porque até o momento não atribuímos nenhum valor para ela. Então quando nenhum valor for atribuído para uma variável e se tentarmos utilizar essa variável, o JavaScript de forma automática vai atribuir `undefined` para essa variável ou seja não dissemos de forma intencional que gostaríamos que essa variável fosse `undefined`, porém o JavaScript inseriu de forma automática esse valor para a variável(`nothingHere`).



/igor-rebolla

Null e Undefined (uma duplinha do barulho - revisão)

Eu vou deixar a revisão de quando o NULL é utilizado e de quando o UNDEFINED é utilizado, com a brilhante explicação que vocês deram durante nossa aula.

"Oh é assim se com o Null o valor é atribuído de forma intencional, então quando é UNDEFINED o nosso terceiro pilar do curso que aqui é o JavaScript(JS), ele atribui de forma automática esse valor em uma variável que ainda não tem nenhum valor guardado nela." Alunos

CLAP👏 CLAP👏 CLAP👏... (Onomatopeia de Palmas para vocês)



Sugestões de prática:

1. Refazer e Entender os exemplos dessa aula;
2. Usar valores definidos por vocês para praticar mais sobre: Null e Undefined;
3. Acrescentar 1 novo console.log, minha sugestão inverter a ordem dentro desse console.log:
De: `console.log(nothingHere + 21)`
Para: `console.log(21 + nothingHere)`
Salvar o arquivo app.js e verificar no console do navegador se o resultado obtido é diferente ou igual visto no exemplo da aula.

Colocar isso em prática no VSCode e ver o resultado no console do Navegador.

Programação é prática, curiosidade e repetição!!!!



Array(s)

Igor, Igor, Igor..... Quando que esse tal de Array(s) será utilizado?

Muito bom, logo no inicio da aula mais uma excelente pergunta, assim como aconteceu na aula passada. Estou gostando disso!!!!

Usamos o array para guardarmos uma lista de valores que usualmente tem uma relação entre si. Podemos guardar uma lista que possui números ou uma lista que possui strings. E exemplificando ainda mais isso: o Array de números (possui os números dos meses do ano) e o Array de strings (possui os nomes dos meses do ano).

E bora ver o primeiro exemplo de Array, no caso aqui um Array de Números.....

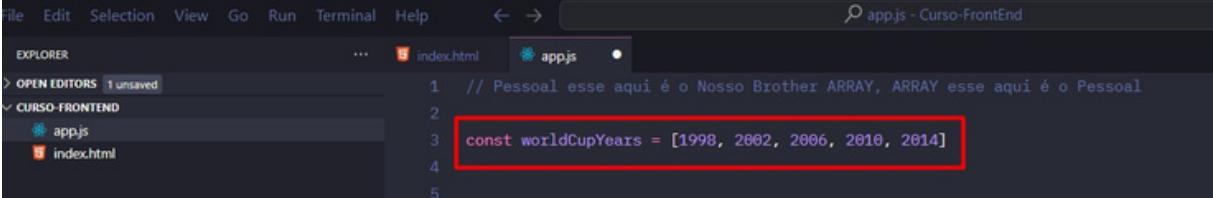
Ah, antes disso..... dentro do nosso ilustríssimo arquivo app.js, na linha 1 criaremos um comentário: // Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal

Então logo abaixo do comentário: // Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal, declararemos uma const worldCupYears que receberá um array [] com os números: 1998, 2002, 2006, 2010, 2014. Nesse exemplo temos um array dos anos que aconteceram a Copa do Mundo. Ficando assim:

```
const worldCupYears = [1998, 2002, 2006, 2010, 2014]
```



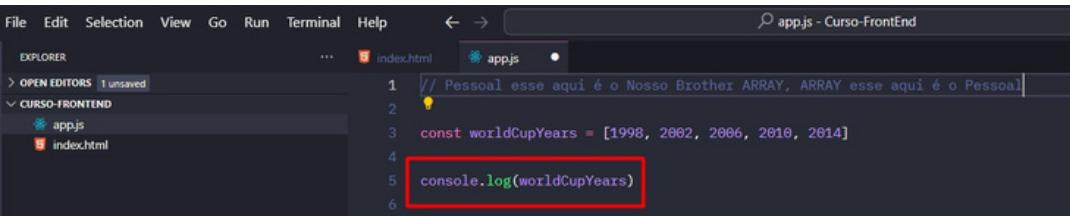
Array(s)



```
File Edit Selection View Go Run Terminal Help ← → ⌘ app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal
2
3 const worldCupYears = [1998, 2002, 2006, 2010, 2014]
4
5
```

Adicionaremos um console.log passando a worldCupYears. Ficando assim:

```
console.log(worldCupYears)
```



```
File Edit Selection View Go Run Terminal Help ← → ⌘ app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal
2
3 const worldCupYears = [1998, 2002, 2006, 2010, 2014]
4
5 console.log(worldCupYears)
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que os 5 itens do array worldCupYears: 1998, 2002, 2006, 2010, 2014 estão sendo exibidos no console.



```
Console Elements Recorder > ⚙️ ⓘ Default levels ▾ No Issues ⚙️
∅ top ⓘ Filter (5) [1998, 2002, 2006, 2010, 2014] app.js:5
```



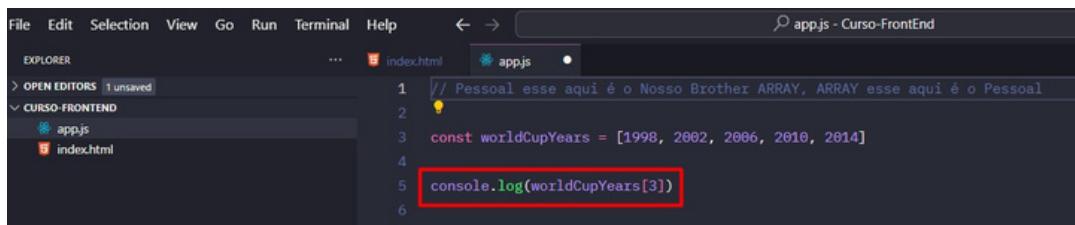
/igor-rebolla

Array(s)

E se nós quisermos que apenas o ano da Copa do Mundo de 2010 seja exibido no console do navegador.

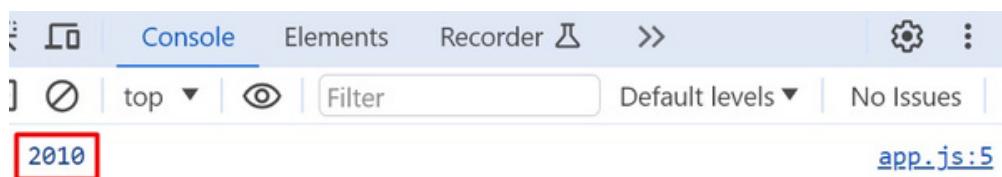
Nesse caso usaremos a notação de colchetes aqui no final da referência do array, ou seja da mesma forma que vimos na aula sobre acessarmos um único caractere de uma string e como fizemos isso... Alguém lembra? Isso mesmo, foi quando passamos o index dela. Por esse motivo aqui dentro especificaremos o index 3. Então aqui no console.log, podemos exibir o quarto item desse array que aqui é o ano de 2010, e dentro do console.log() vamos inserir a worldCupYears index 3. Ficando assim:

```
console.log(worldCupYears[3])
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved
CURSO-FRONTEND app.js
index.html
app.js
1 // Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal
2
3 const worldCupYears = [1998, 2002, 2006, 2010, 2014]
4
5 console.log(worldCupYears[3])
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o quarto item está do array sendo exibido no console que aqui no caso é o: 2010.



/igor-rebolla

Array(s)

Até aqui trabalhamos com Array que possui apenas números, agora veremos um exemplo de um Array que possui apenas strings.

Como nós já criamos dentro do arquivo app.js o nosso comentário: // Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal, vamos dar aquele enter maroto para quebrarmos uma linha e na linha 3, declararemos uma let worldCupChampions que receberá um array abrindo e fechando colchetes [] essa é a notação de um array, e agora dentro desse array teremos 5 strings com o nome de cinco seleções distintas que venceram a Copa do Mundo, adicionaremos a primeira seleção 'France' agora bora adicionar uma vírgula e adicionaremos uma segunda seleção 'Brazil' bora adicionar outra vírgula aqui e adicionaremos uma terceira seleção 'Italy' bora adicionar uma vírgula aqui também e adicionaremos uma quarta seleção 'Spain' bora adicionar mais uma vírgula aqui (calma pessoal que tá acabando) e adicionaremos uma quinta e última seleção 'Germany' e não precisamos mais adicionar uma vírgula aqui (viu, eu disse que tava acabando) depois do último item do array.

Passo a passo do que foi feito no texto acima:

- 1 - Criamos uma let chamada worldCupChampions
- 2 - Colocamos o sinal de = que aqui não quer dizer igual e sim atribuição
- 3 - Depois do sinal de atribuição colocamos a notação de array que é representado pela abertura e fechamento de colchetes []
- 4 - Definimos que teremos 5 strings com o nome de cinco seleções distintas que venceram a Copa do Mundo
- 5 - Abrimos os colchetes [e colocamos a nossa primeira seleção que é a: ['France']



/igor-rebolla

Array(s)

Continuação do Passo a passo do que foi feito na página anterior:

- 6 - Colocamos uma vírgula logo após a string 'France', ficando assim:
['France',
- 7 - Dentro dos parênteses, vamos colocar a nossa segunda seleção que é o:
Brazil, logo após a vírgula da string 'France', ficando assim ['France',
'Brazil']
- 8 - Colocamos uma vírgula logo após a string 'Brazil', ficando assim:
['France', 'Brazil']
- 9 - Dentro dos parênteses, vamos colocar a nossa Terceira seleção que é a:
Italy, logo após a vírgula da string 'Brazil', ficando assim ['France',
'Brazil', 'Italy']
- 10 - Colocamos uma vírgula logo após a string 'Italy', ficando assim:
['France', 'Brazil', 'Italy']
- 11 - Dentro dos parênteses, vamos colocar a nossa Quarta seleção que é a:
Spain, logo após a vírgula da string 'Italy', ficando assim ['France',
'Brazil', 'Italy', 'Spain']
- 12 - Colocamos uma vírgula logo após a string 'Spain', ficando assim:
['France', 'Brazil', 'Italy', 'Spain']
- 13 - Dentro dos parênteses, vamos colocar a nossa Quinta seleção que é a:
Germany, logo após a vírgula da string 'Spain', ficando assim ['France',
'Brazil', 'Italy', 'Spain', 'Germany']
- 14 - Como Germany é o último item do nosso array não precisamos
adicionar uma vírgula aqui, basta fecharmos o parêntese] que o nosso
array estará pronto, ficando assim:
['France', 'Brazil', 'Italy', 'Spain', 'Germany']
- 15 - Agora juntamos o nosso array com a let que criamos, ficando assim:
`let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']`



/igor-rebolla

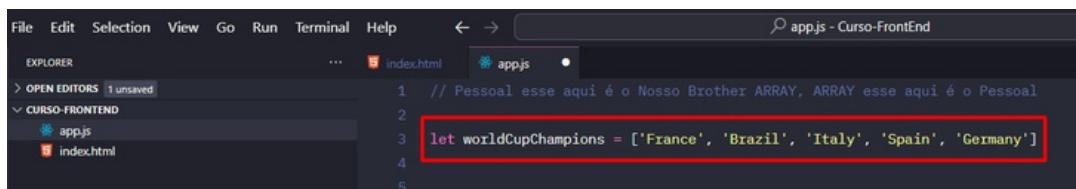
Array(s)

Igor, Igor, Igor... porque você não explicou a notação de colchetes e nem detalhou passo a passo no primeiro exemplo dos números???

Igor Responde: foi para que vocês fizessem do jeito que está lá (tentando entender o que estava acontecendo) e caso tivessem alguma dúvida, quando chegassem aqui (vocês associassem esse exemplo aqui e replicassem para os números).

Vimos aqui um array que possui 5 itens dentro dele, guardamos esses 5 valores dentro de uma só variável a qual chamamos de worldCupChampions. E já nesse segundo exemplo visualizamos o quanto o nosso brother array é útil (no nosso caso aqui ele armazenou uma lista de dados (5 seleções) que tem relação entre si. Ficando assim:

```
let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
```



A screenshot of a code editor interface, likely VS Code. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar for 'app.js - Curso-FrontEnd'. The Explorer sidebar shows 'OPEN EDITORS' with 1 unsaved file and a folder 'CURSO-FRONTEND' containing 'app.js' and 'index.html'. The main editor area has five lines of code. Line 1 starts with a comment: '// Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal'. Lines 2 and 3 contain the declaration: 'let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']'. Lines 4 and 5 are blank. A red rectangular box highlights the entire declaration on line 3.

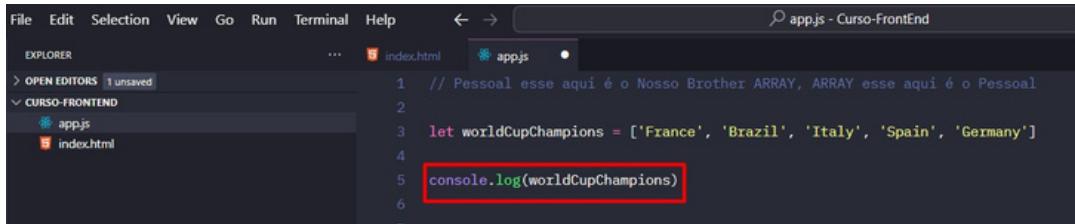
```
// Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal
let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
```

Adicionaremos um console.log passando a worldCupChampions. Ficando assim:

```
console.log(worldCupChampions)
```



Array(s)



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved index.html app.js
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal
2
3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
4
5 console.log(worldCupChampions)
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que é exibido esse ARRAY de CAMPEÕES DA COPA DO MUNDO, com:

(5) ['France', 'Brazil', 'Italy', 'Spain', 'Germany']



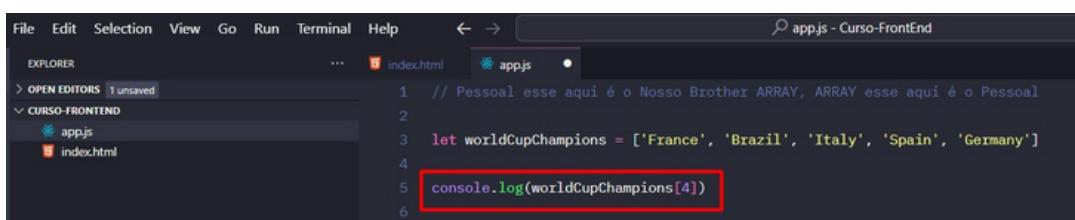
```
Console Elements Recorder > ➡️
top ▾ Filter Default levels ▾ No Issues
(5) ['France', 'Brazil', 'Italy', 'Spain', 'Germany'] app.
```

Array(s)

Agora vem a pergunta do milhão... brincadeira é sobre array mesmo.

E se quisermos obter assim como quem não quer nada, apenas 1 desses itens (NO CASO AQUI - GERMANY), nesse caso usaremos a notação de colchetes aqui no final da referência do array, ou seja da mesma forma vimos na aula sobre acessarmos um único caractere de uma string e como fizemos isso... Alguém lembra? Isso mesmo, foi quando passamos o index dela e aqui dentro especificaremos o index 4. Ficando assim:

```
console.log(worldCupChampions[4])
```



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal
2
3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
4
5 console.log(worldCupChampions[4])
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome), que o quarto item do array foi exibido no console que no caso aqui é a Germany.



/igor-rebolla

Array(s)

Igor, Igor, Igor... Pode por favor recapitular conosco: O que essa expressão `worldCupChampions[4]` está fazendo?

Pra já.... nós queremos obter o item que está no index 4 desse array `worldCupChampions` e como o JavaScript(JS) é baseado no zero(zero-based) ou seja ele conta esses itens com Zero (0). Então no nosso exemplo aqui, vai ficar assim: France (0) que no nosso exemplo aqui é o Brazil (1) que no nosso exemplo aqui é a Italy (2) que no nosso exemplo aqui é a Spain (3) que no nosso exemplo aqui é a Germany (4). Então essa expressão `worldCupChampions[4]` ele retornará o quinto item do Array.

Para um melhor entendimento, os itens do texto acima, são representados dessa forma dentro de um array (por conta do zero-based). Conforme exemplo abaixo:

Index 0 do array que faz referência ao Item 1 que no caso aqui é France
Index 1 do array que faz referência ao Item 2 que no caso aqui é Brazil
Index 2 do array que faz referência ao Item 3 que no caso aqui é Italy
Index 3 do array que faz referência ao Item 4 que no caso aqui é Spain
Index 4 do array que faz referência ao Item 5 que no caso aqui é Germany



Array(s)

Agora eu faço uma pergunta marota para vocês.... Nós acabamos de ver 2 exemplos de Arrays: O primeiro exemplo um Array com Números e o Segundo exemplo um Array com Strings.

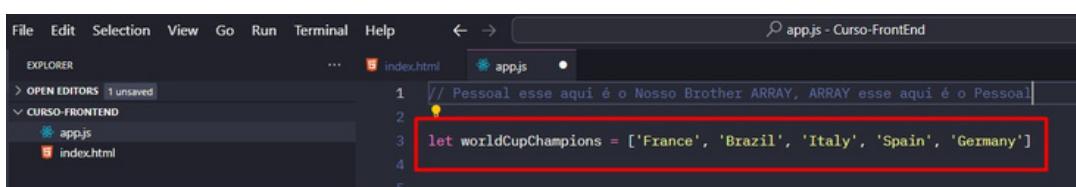
Será que conseguimos armazenar um Array que contem uma “Mistura” de Números e Strings???

Eita, estamos em dúvidas aqui, acreditamos que sim, mas poderia mostrar um exemplo para melhor entendimento.

É pra já bora ver isso então:

Como nós tentaremos fazer uma “Mistura” dos arrays de números e Strings, manteremos a `let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']`. Ficando assim:

```
let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
```



The screenshot shows a code editor interface with a dark theme. At the top, there's a menu bar with File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar labeled 'app.js - Curso-FrontEnd'. Below the menu is an 'EXPLORER' sidebar showing 'OPEN EDITORS 1 unsaved' and a folder 'CURSO-FRONTEND' containing 'app.js' and 'index.html'. The main workspace contains the following code:

```
// Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal
let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
```

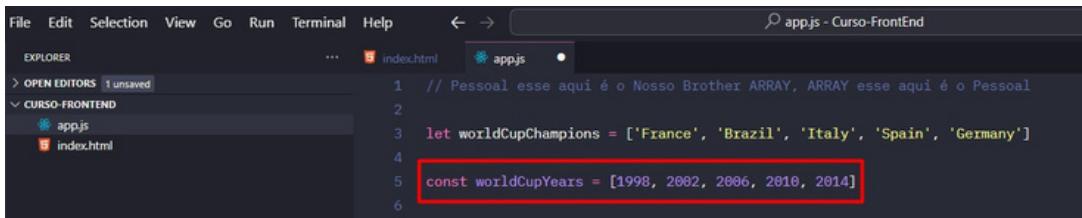
The third line of code, which defines the array, is highlighted with a red rectangle.

E manteremos também a `const worldCupYears = [1998, 2002, 2006, 2010, 2014]`. Ficando assim:

```
const worldCupYears = [1998, 2002, 2006, 2010, 2014]
```



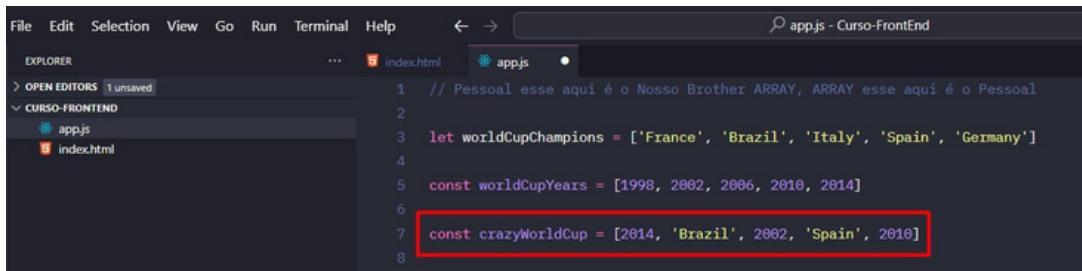
Array(s)



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
> OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal
2
3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
4
5 const worldCupYears = [1998, 2002, 2006, 2010, 2014]
6
```

Agora que tanto a `let worldCupChampions`, quanto a `const worldCupYears` foram mantidas, criaremos uma `const` que guardará essa "Mistura Maluquinha", declararemos uma `const crazyWorldCup` que receberá um array [] com um `number: 2014`, como primeiro item e depois como segundo item receberá: 'Brazil', como terceiro item receberá: 2002, como quarto item receberá: 'Spain' e como quinto item receberá: 2010. Notem que alternamos entre strings e números. Ficando assim:

```
const crazyWorldCup = [2014, 'Brazil', 2002, 'Spain', 2010]
```



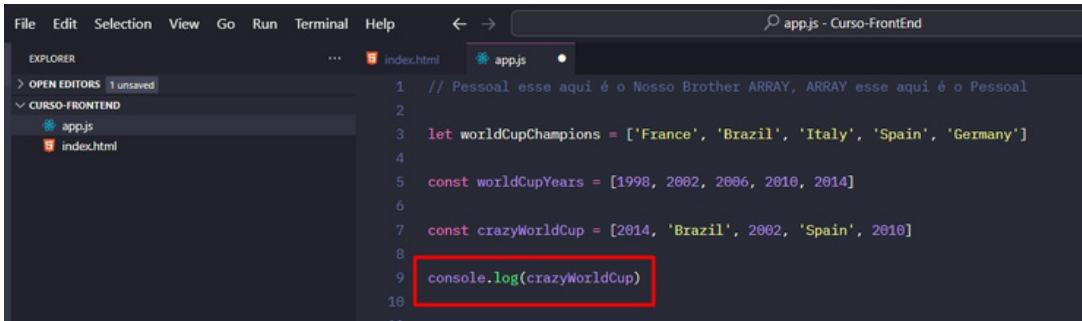
```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
> OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal
2
3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
4
5 const worldCupYears = [1998, 2002, 2006, 2010, 2014]
6
7 const crazyWorldCup = [2014, 'Brazil', 2002, 'Spain', 2010]
8
```

Adicionaremos um `console.log` passando a `crazyWorldCup`. Ficando assim:

```
console.log(crazyWorldCup)
```



Array(s)



```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui é o Nosso Brother ARRAY, ARRAY esse aqui é o Pessoal
2
3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
4
5 const worldCupYears = [1998, 2002, 2006, 2010, 2014]
6
7 const crazyWorldCup = [2014, 'Brazil', 2002, 'Spain', 2010]
8
9 console.log(crazyWorldCup)
10
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o array crazyWorldCup [2014, 'Brazil', 2002, 'Spain', 2010] é exibido no console.



O que eu queria mostrar para vocês aqui é que é sim possível armazenarmos 2 tipos diferentes de dados (números e strings) em apenas 1 único array.

Notem que isso parece não fazer muito sentido, até por que a definição de array a qual vimos na página 2 dessa aula, diz que: Usamos o array para guardarmos uma lista de valores que usualmente tem uma relação entre si. Podemos guardar uma lista que contém números ou uma lista que contém strings.

Array é PUSH, Array é Tech - (Método PUSH)

Veremos agora o nosso primeiro Método de Array que é o: PUSH

O PUSH ele vai incluir elementos no fim do Array e vai retornar o seu comprimento (length).

Aqui criaremos um comentário // Array também é PUSH, Array também é Tech - Pessoal esse aqui é o Método PUSH, Método PUSH esse aqui é o Pessoal.

E logo abaixo desse comentário, manteremos a nossa ilustríssima let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']. Ficando assim:



```
File Edit Selection View Go Run Terminal Help app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved CURSO-FRONTEND app.js index.html
1 // Array também é PUSH, Array também é Tech - Pessoal esse aqui é o Método PUSH, Método PUSH esse aqui é o Pessoal
2
3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
4
```

The screenshot shows a code editor interface with a dark theme. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar says "app.js - Curso-FrontEnd". The left sidebar shows an "EXPLORER" section with "OPEN EDITORS 1 unsaved" and a "CURSO-FRONTEND" folder containing "app.js" and "index.html". The main code editor area has four lines of code. Line 1 is a multi-line comment starting with "//". Line 3 contains the declaration "let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']". The line numbers 1, 2, 3, and 4 are visible on the left. A red rectangular box highlights the entire line 3 code.

Daremos aquele enter maroto para quebrarmos uma linha, e na linha 5 do nosso arquivo app.js, criaremos uma let moreTwoWorldCupChampions = worldCupChampions.push ('Argentina', 'Uruguay').

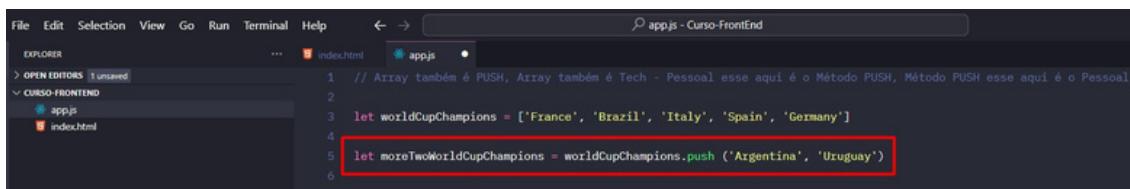
Igor, Igor, Igor..... Tá confuso.. pode explicar melhor isso???

Posso sim: Ao declararmos a let moreTwoWorldCupChampions ela receberá(já com o método push()) a worldCupChampions.push() e aqui dentro dos parênteses do push() os novos valores que poderemos empurrar (push) lá paraaaaaaaaaaaa o final desse array, então no nosso exemplo passaremos ('Argentina', 'Uruguay'). Ficando assim:



Array é PUSH, Array é Tech - (Método PUSH)

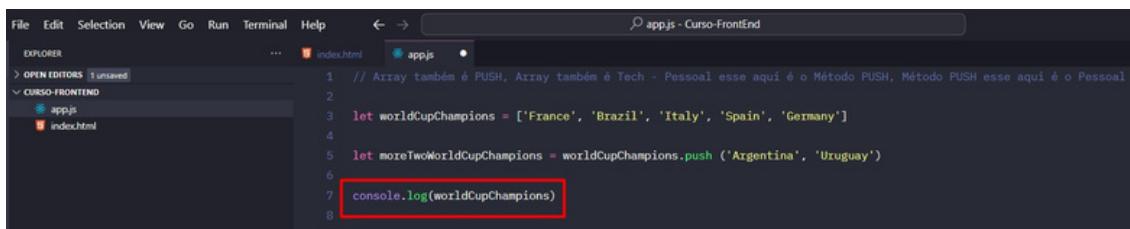
```
let moreTwoWorldCupChampions = worldCupChampions.push ('Argentina', 'Uruguay')
```



```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
index.html
1 // Array também é PUSH, Array também é Tech - Pessoal esse aqui é o Método PUSH, Método PUSH esse aqui é o Pessoal
2
3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
4
5 let moreTwoWorldCupChampions = worldCupChampions.push ('Argentina', 'Uruguay')
6
```

Adicionaremos um `console.log` passando a `worldCupChampions`. Ficando assim:

```
console.log(worldCupChampions)
```



```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
index.html
index.html
1 // Array também é PUSH, Array também é Tech - Pessoal esse aqui é o Método PUSH, Método PUSH esse aqui é o Pessoal
2
3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
4
5 let moreTwoWorldCupChampions = worldCupChampions.push ('Argentina', 'Uruguay')
6
7 console.log(worldCupChampions)
8
```

E ao salvarmos o arquivo `app.js` com aquele `CTRL + S` maroto podemos ver no `console` do Navegador(Chrome) que ela agora tem os 2 novos items (Argentina e Uruguay) que adicionamos, sendo assim ela agora possui 7 itens.



Array é PUSH, Array é Tech - (Método PUSH)



The screenshot shows a browser's developer tools console tab. At the top, there are tabs for 'Console' (which is selected), 'Elements', 'Recorder', and 'Default levels'. Below the tabs are buttons for 'top', 'Filter', 'Default levels', and 'No Issues'. The main area of the console displays the following code and its output:

```
▶ (7) ['France', 'Brazil', 'Italy', 'Spain', 'Germany', 'Argentina',  
      'Uruguay']
```

The output is highlighted with a yellow box. The file path 'app.js:10' is visible above the output. The numbers '(7)' indicate the length of the array.



/igor-rebolla

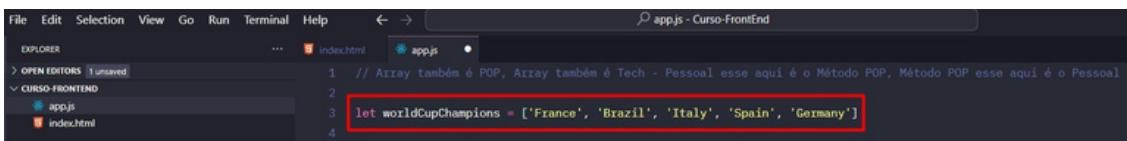
Array é POP, Array é Tech - (Método POP)

Veremos agora o nosso segundo Método de Array que é o: POP

O POP ele vai remover o último elemento do Array, retornando esse elemento.

Aqui criaremos um comentário // Array também é POP, Array também é Tech - Pessoal esse aqui é o Método POP, Método POP esse aqui é o Pessoal, e logo abaixo desse comentário manteremos a nossa let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']. Ficando assim:

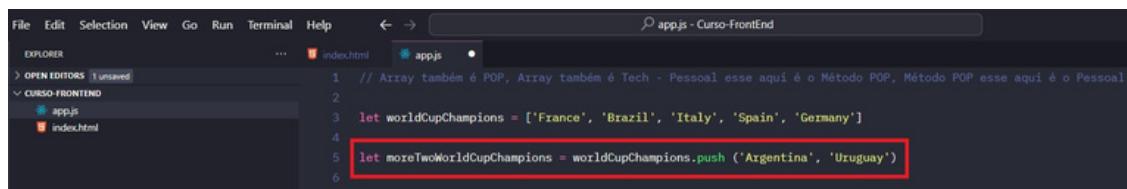
```
let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
```



A screenshot of the Visual Studio Code interface. The menu bar shows 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The title bar says 'app.js - Curso-FrontEnd'. The Explorer sidebar shows 'OPEN EDITORS' with 'index.html' and 'app.js' listed. The code editor has four lines of code: 1 // Array também é POP, Array também é Tech - Pessoal esse aqui é o Método POP, Método POP esse aqui é o Pessoal, 2, 3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany'], and 4. The line 'let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']' is highlighted with a red rectangle.

Manteremos também a let moreTwoWorldCupChampions = worldCupChampions.push ('Argentina', 'Uruguay'). Ficando assim:

```
let moreTwoWorldCupChampions = worldCupChampions.push ('Argentina',  
'Uruguay')
```



A screenshot of the Visual Studio Code interface, similar to the previous one. The code editor now has six lines of code: 1 // Array também é POP, Array também é Tech - Pessoal esse aqui é o Método POP, Método POP esse aqui é o Pessoal, 2, 3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany'], 4, 5 let moreTwoWorldCupChampions = worldCupChampions.push ('Argentina', 'Uruguay'), and 6. The line 'let moreTwoWorldCupChampions = worldCupChampions.push ('Argentina', 'Uruguay')' is highlighted with a red rectangle.

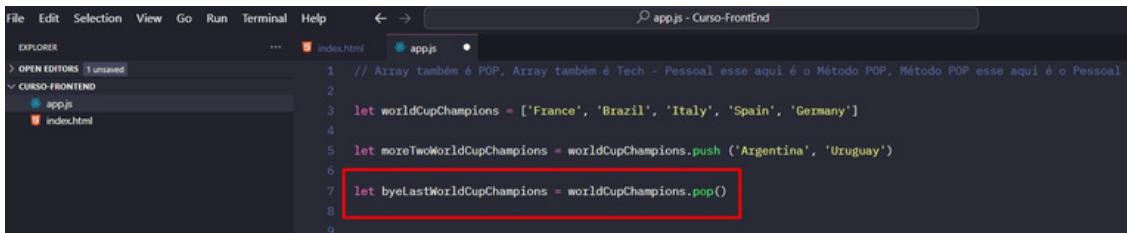


/igor-rebolla

Array é POP, Array é Tech - (Método POP)

E declararemos + uma let byeLastWorldCupChampions que receberá worldCupChampions.pop(). Ficando assim:

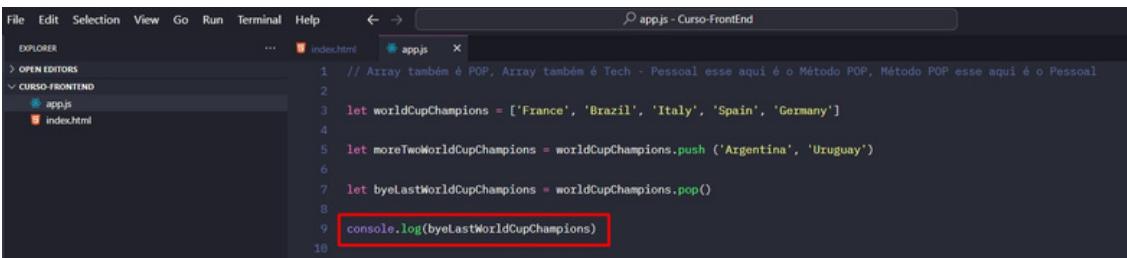
```
let byeLastWorldCupChampions = worldCupChampions.pop()
```



```
File Edit Selection View Go Run Terminal Help ← → ⌘ appjs - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved
CURSO-FRONTEND app.js index.html
1 // Array também é POP, Array também é Tech - Pessoal esse aqui é o Método POP, Método POP esse aqui é o Pessoal
2
3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
4
5 let moreTwoWorldCupChampions = worldCupChampions.push ('Argentina', 'Uruguay')
6
7 let byeLastWorldCupChampions = worldCupChampions.pop()
8
9
```

Adicionaremos um console.log passando a byeLastWorldCupChampions. Ficando assim:

```
console.log(byeLastWorldCupChampions)
```



```
File Edit Selection View Go Run Terminal Help ← → ⌘ appjs - Curso-FrontEnd
EXPLORER OPEN EDITORS CURSO-FRONTEND app.js index.html
1 // Array também é POP, Array também é Tech - Pessoal esse aqui é o Método POP, Método POP esse aqui é o Pessoal
2
3 let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
4
5 let moreTwoWorldCupChampions = worldCupChampions.push ('Argentina', 'Uruguay')
6
7 let byeLastWorldCupChampions = worldCupChampions.pop()
8
9 console.log(byeLastWorldCupChampions)
10
11
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que Uruguay foi exibido



Array é POP, Array é Tech - (Método POP)



Igor, Igor, Igor, Igor..... E porque Uruguay apareceu ai?
Porque esse método POP vai remover o último item de um array(Uruguay).
E retornará esse item que no caso aqui é Uruguay

E para comprovar isso, adicionaremos um console.log passando a worldCupChampions. Ficando assim:

```
console.log(worldCupChampions)
```

A screenshot of a code editor showing the 'app.js' file. The file contains the following code:

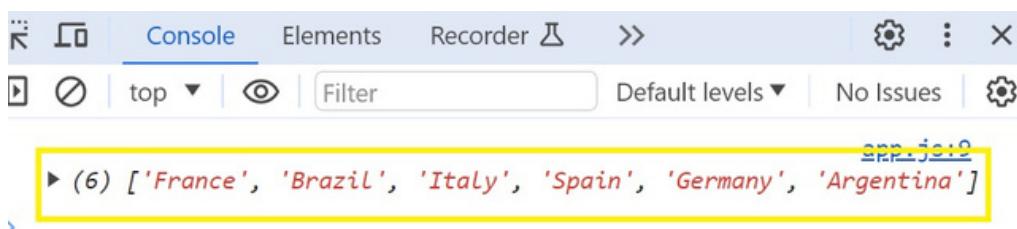
```
// Array também é POP, Array também é Tech - Pessoal esse aqui é o Método POP, Método POP esse aqui é o Pessoal
let worldCupChampions = ['France', 'Brazil', 'Italy', 'Spain', 'Germany']
let moreTwoWorldCupChampions = worldCupChampions.push('Argentina', 'Uruguay')
let byeLastWorldCupChampions = worldCupChampions.pop()
console.log(worldCupChampions)
```

A red rectangular box highlights the line 'console.log(worldCupChampions)'.

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o Uruguay não é mais o último item desse array.



Array é POP, Array é Tech - (Método POP)



The screenshot shows the Chrome DevTools Console tab. At the top, there are tabs for 'Console' (which is selected), 'Elements', 'Recorder', and other developer tools. Below the tabs are buttons for 'top' (with a dropdown arrow), 'Filter' (with a search input), 'Default levels' (with a dropdown arrow), 'No Issues' (with a gear icon), and another gear icon. The main area displays a list of objects. The first item in the list is highlighted with a yellow border and has a blue arrow pointing to it from the left. The object is represented by a red string: '▶ (6) ['France', 'Brazil', 'Italy', 'Spain', 'Germany', 'Argentina']'. Above this list item, the text 'app.js:2' is visible.



/igor-rebolla

Sugestões de prática:

1. Refazer e Entender os exemplos dessa aula;
2. Usar valores definidos por vocês para praticar mais sobre Array(s): Números e Strings;
3. Usar valores definidos por vocês para praticar mais sobre Métodos de Array(s): Push e Pop
4. Quando misturamos 2 array(s) diferentes (números e strings) notem que usamos uma let para armazenar os campeões da copa do mundo (worldCupChampions) e uma const pra armazenar os anos que aconteceram a copa do mundo (worldCupYears). E se tivéssemos usados 2 consts ou 2 lets o resultado teria sido o mesmo? Fica mais essa sugestão para vocês realizarem!!!

Colocar isso em prática no VSCode e ver o resultado no console do Navegador.

Programação é prática, curiosidade e repetição!!!!



Booleans

Igor, Igor, Igor..... Esse termo Booleans para quem está começando e/ou migrando para essa área de Desenvolvimento é uma novidade, então poderia explicar com exemplos para visualizarmos também na prática como o Booleans funciona.

Show de bola, esse termo Booleans pode parecer algo de “outro mundo” em um primeiro contato, mas explicaremos de um jeito lúdico para que vocês possam entender como ele funciona. Mais uma vez, um ótimo questionamento. Estou gostando muitoooooooooooo disso!!!!

Então bora lá, ver esse tal de Booleans...

Booleans nos mostra 2 valores no JavaScript(JS): False or True (Falso ou Verdadeiro). É só isso que nós precisamos saber nesse momento. Conforme o decorrer dessa aula, veremos mais detalhes sobre Booleans com o auxílio dos exemplos.

E bora ver o primeiro exemplo de Booleans.....

Ah, antes disso..... dentro do nosso nobre arquivo app.js, na linha 1 criaremos um comentário: // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal

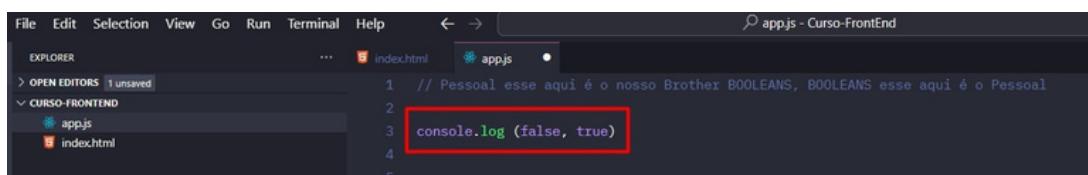
Então logo abaixo do comentário: // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal, especificaremos um console.log e passaremos como argumento um false e um true (só para esquentarmos os nossos motores). Observem que esse false e esse true podem confundir com strings, só que esses 2 valores não são considerados strings.



Booleans

Alguém, lembra quando consideramos uma string?? Isso mesmo, para que seja considerada uma string nós precisaríamos envolver tanto false quanto true entre aspas (Sejam elas simples ou duplas). Então veremos como o false e o true sem aspas, serão declarados no console.log. Ficando assim:

console.log (false, true)



```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 console.log (false, true)
4
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que false true são mostrados.



Bora exemplificar melhor aqui (para entendermos quando é uma string com aspas e quando não é sem aspas). No Primeiro argumento parti inserir um false, no segundo argumento parti inserir um true, no terceiro argumento parti inserir uma string 'false', no quarto argumento parti inserir uma string 'true', no quinto argumento parti inserir um true, no sexto elemento parti inserir um false, no sétimo elemento parti inserir uma string 'true', e no oitavo elemento parti inserir uma string 'false' .

Bora ver na próxima página, o Passo a passo do que foi feito no texto acima:



Booleans

- 1 - Definimos que vamos usar o `console.log()`
- 2 - Abriremos o parênteses (logo após o `console.log()`)
- 3 - Dentro dessa abertura dos parênteses, vamos inserir o nosso primeiro argumento que é `false`. Ficando assim: `console.log(false)`
- 4 - Depois do `false`, vamos inserir uma vírgula , e daremos um espaço. Ficando assim: `console.log(false,`
- 5 - Com o espaço dado, vamos inserir o nosso segundo argumento que é `true`. Ficando assim: `console.log(false, true)`
- 6 - Depois do `true`, vamos inserir uma vírgula , e daremos um espaço. Ficando assim: `console.log(false, true,`
- 7 - Com o espaço dado, vamos inserir o nosso terceiro argumento que é uma string '`false`'. Ficando assim: `console.log(false, true, 'false')`
- 8 - Depois da string '`false`', vamos inserir uma vírgula, e daremos um espaço. Ficando assim: `console.log(false, true, 'false',`
- 9- Com o espaço dado, vamos inserir o nosso quarto argumento que é uma string '`true`'. Ficando assim: `console.log(false, true, 'false', 'true')`
- 10 - Depois da string '`true`', vamos inserir uma vírgula, e daremos um espaço. Ficando assim: `console.log(false, true, 'false', 'true',`
- 11 - Com o espaço dado, vamos inserir o nosso quinto argumento que é o `true`. Ficando assim: `console.log(false, true, 'false', 'true', true)`
- 12- Depois do `true`, vamos inserir uma vírgula , e daremos um espaço. Ficando assim: `console.log(false, true, 'false', 'true', true,`
- 13 - Com o espaço dado, vamos inserir o nosso sexto argumento que é o `false`. Ficando assim: `console.log(false, true, 'false', 'true', true, false)`
- 14 - Depois do `false`, vamos inserir uma vírgula , e daremos um espaço. Ficando assim: `console.log(false, true, 'false', 'true', true, false,`



/igor-rebolla

Booleans

Continuação do Passo a passo do que foi feito na página anterior:

- 15 - Com o espaço dado, vamos inserir o nosso sétimo argumento que é a string 'true'. Ficando assim: console.log(false, true, 'false', 'true', true, false, 'true')
- 16 - Depois da string 'true', vamos inserir uma vírgula, e daremos um espaço. Ficando assim: console.log(false, true, 'false', 'true', true, false, 'true',)
- 17 - Com o espaço dado, vamos inserir o nosso oitavo argumento que é a string 'false'. Ficando assim: console.log(false, true, 'false', 'true', true, false, 'true', 'false')
- 18 - Como a string 'false' é o último argumento dentro dos parênteses do console.log... não precisamos adicionarmos uma vírgula aqui, basta fecharmos os parênteses) que o nosso console.log estará pronto. Ficando assim:

```
console.log (false, true, 'false', 'true', true, false, 'true', 'false')
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que false, true, 'false', 'true', true, false, 'true', 'false' foram mostrados.



/igor-rebolla

Booleans

Resuminho Maroto do Exemplo Acima:

O primeiro valor(1°), o segundo valor(2°), o quinto valor(5°) e o sexto valor(6°) passados como argumentos, são diferentes do terceiro valor(3°), do quarto valor(4°), do sétimo valor(7°) e do oitavo valor(8°) passados como argumentos, pois tanto o `false`(1°) e o `true`(2°), quanto o `true`(5°) e o `false`(6°) são booleans não possuem as aspas, já o '`false`'(3°), o '`true`'(4°), o '`true`'(7°) e o '`false`'(8°) são strings que possuem as aspas.

Podemos observar nesse exemplo que o console do Navegador(Chrome) foi brother demais conosco ele inseriu de forma amigável: aspas nas strings, pra que pudéssemos “bater os olhos” e notarmos a diferença entre esses valores.

Igor, Igor, Igor... belezinha ficou mais claro depois dessa explicação e dos exemplos, poderia fazer um resumo.... E quando nós usaremos Booleans?
Só se for agora:

Quando precisarmos verificar uma condição dentro do código que estamos desenvolvendo ou seja vão existir ocasiões que será necessário verificarmos se um bloco de código é `false` ou `true` ou seja falso ou verdadeiro.

E pra tornar esse “paranauê” de Booleans ainda mais divertido... podemos utilizar métodos que nos mostram um Boolean que por sua vez retornará um `false` ou um `true`. Podemos utilizar métodos que nos devolvem um Boolean (`false` ou `true`). E na próxima página veremos, esse método que é o: `Includes`

Partiu ver sobre `Includes`... uhuuuuu estou animado pra vermos isso!!!



Método Includes()

Igor, Igor, Igor, Igor, Igor... E esse tal de método Includes, o que seria?

Igor, responde: O tal do método includes() vai nos informar se um array/objeto possui um determinado item ou não, retornando false ou true.

A explicação acima foi o resumo do resumo do que é o Método Includes, só para vocês terem uma ideia do que falaremos, pois com os exemplos na sequência, o entendimento ficará mais claro e depois do primeiro exemplo, passaremos um Resumo Maroto do que é o Includes em parceria com o exemplo que foi visto.

Pode ser pessoal? Pode sim, Igor!!!

Uhulllll....Então bora ver o primeiro exemplo do Includes...

Como nós já criamos dentro do arquivo app.js o nosso comentário // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal, daremos um enter para quebrarmos uma linha e na linha 3, declararemos uma let houseRentValue que recebe uma string 'R\$ 1000 - É pra alugar hoje'. Ficando assim:

```
let houseRentValue = 'R$ 1000 - É pra alugar hoje'
```



Método Includes()



```
File Edit Selection View Go Run Terminal Help ⏪ ⏩ app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let houseRentValue = 'R$ 1000 - É pra alugar hoje'
4
5
```

Declaremos uma let hasDollarSign que recebe houseRentValue.includes() e que vai receber dentro dos parênteses () por argumento uma string '\$'. Ficando assim:

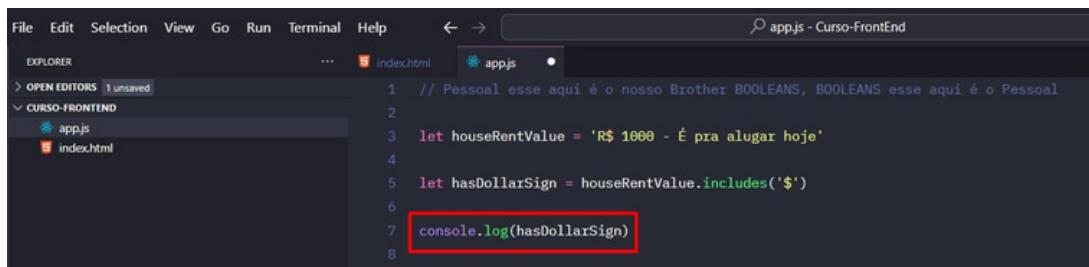
```
let hasDollarSign = houseRentValue.includes('$')
```



```
File Edit Selection View Go Run Terminal Help ⏪ ⏩ app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let houseRentValue = 'R$ 1000 - É pra alugar hoje'
4
5 let hasDollarSign = houseRentValue.includes('$')
6
```

Adicionaremos um console.log passando a hasDollarSign. Ficando assim:

```
console.log(hasDollarSign)
```



```
File Edit Selection View Go Run Terminal Help ⏪ ⏩ app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let houseRentValue = 'R$ 1000 - É pra alugar hoje'
4
5 let hasDollarSign = houseRentValue.includes('$')
6
7 console.log(hasDollarSign)
8
```



/igor-rebolla

Método Includes()

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que true está sendo mostrado.



Hora do Resumo Maroto:

O que é o tal do Método Includes em Parceria com o exemplo visto acima.

O ilustríssimo método verificará se uma string '\$' que utilizamos no nosso exemplo, se ela(string) consta dentro da nossa string: houseRentValue. Se essa string '\$' fizer parte ou seja se ela estiver dentro da string houseRentValue, a expressão houseRentValue.includes('\$') retornará true, agora se essa string '\$' não estiver dentro da string houseRentValue, essa expressão houseRentValue.includes('\$') retornará false.

Fim do Resumo Maroto!!!

No Resumo Maroto nós usamos o método includes para verificarmos quando consta um valor, no nosso exemplo utilizamos o '\$'.

Agora será que é possível fazermos algo para verificarmos caso não tenha o valor dentro da string?? Fica a pergunta do amigo Internauta - Igor (No caso eu mesmo)!!!

Bora ver isso na próxima página....



Método Includes()

Antes manteremos sem nenhuma alteração a nossa primeira let houseRentValue = 'R\$ 1000 - É pra alugar hoje'. Ficando assim:

```
let houseRentValue = 'R$ 1000 - É pra alugar hoje'
```



A screenshot of a code editor window titled 'app.js - Curso-FrontEnd'. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The Explorer sidebar shows 'OPEN EDITORS 1 unsaved' and 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main editor area contains five lines of code:
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let houseRentValue = 'R\$ 1000 - É pra alugar hoje'
4
5

E na nossa segunda let, passaremos a string '%', como argumento dentro do método includes(). Ficando assim:

```
let hasDollarSign = houseRentValue.includes('%')
```



A screenshot of a code editor window titled 'app.js - Curso-FrontEnd'. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The Explorer sidebar shows 'OPEN EDITORS 1 unsaved' and 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main editor area contains six lines of code:
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let houseRentValue = 'R\$ 1000 - É pra alugar hoje'
4
5 let hasDollarSign = houseRentValue.includes('%')

Adicionaremos um console.log passando a hasDollarSign. Ficando assim:

```
console.log(hasDollarSign)
```



Método Includes()



```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html

1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let houseRentValue = 'R$ 1000 - É pra alugar hoje'
4
5 let hasDollarSign = houseRentValue.includes('%')
6
7 console.log(hasDollarSign)
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que false está sendo mostrado.



Igor, Igor, Igor.... E porque false foi mostrado no console do navegador? Por que não temos nenhum caractere '%' dentro da nossa ilustríssima string houseRentValue.

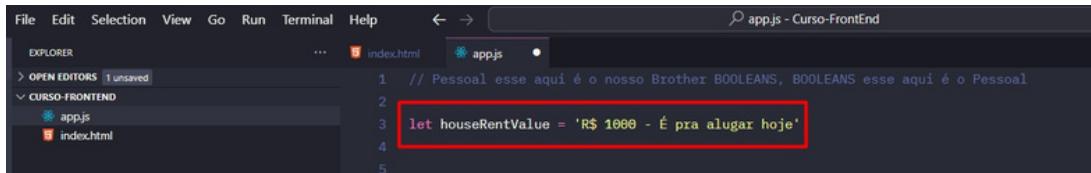
Ah.... até o momento aprendemos como usamos o método includes para fazer aquela verificação marota (toc toc, tem algum caractere ai?) se existe ou não apenas um caractere dentro da string.

Quem lembra o conceito de Strings? Isso mesmo como uma string é um conjunto de caracteres um do lado do outro(sequência), então isso nos permite passarmos seja uma frase completa ou uma única palavra como argumento do includes. Eita, que o negócio tá ficando interessante!!!!



Método Includes()

Antes de vermos esse exemplo, vamos manter a nossa let houseRentValue = 'R\$ 1000 - É pra alugar hoje'. Ficando assim:

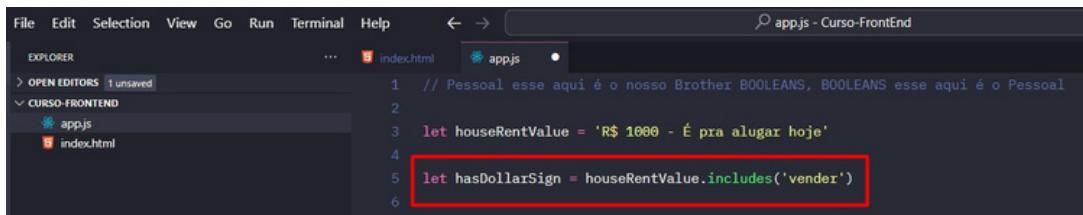


```
File Edit Selection View Go Run Terminal Help ↵ → app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
```

```
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let houseRentValue = 'R$ 1000 - É pra alugar hoje'
4
5
```

E na nossa segunda let, passaremos a string 'vender', como argumento dentro do método includes(). Ficando assim:

```
let hasDollarSign = houseRentValue.includes('vender')
```



```
File Edit Selection View Go Run Terminal Help ↵ → app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
```

```
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let houseRentValue = 'R$ 1000 - É pra alugar hoje'
4
5 let hasDollarSign = houseRentValue.includes('vender')
6
```

Adicionaremos um console.log passando a hasDollarSign. Ficando assim:

```
console.log(hasDollarSign)
```



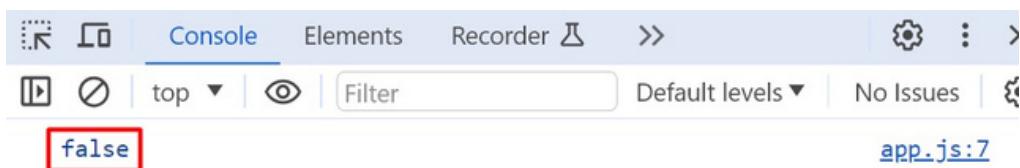
```
File Edit Selection View Go Run Terminal Help ↵ → app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
```

```
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let houseRentValue = 'R$ 1000 - É pra alugar hoje'
4
5 let hasDollarSign = houseRentValue.includes('vender')
6
7 console.log(hasDollarSign)
8
```



Método Includes()

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que false está sendo mostrado.



E se quisermos (assim como quem não quer nada) que o valor da string seja retornado como true. Bora ver isso agora...

Antes de vermos esse exemplo, vamos manter a nossa let houseRentValue = 'R\$ 1000 - É pra alugar hoje'. Ficando assim:



E na nossa segunda let, passaremos a string 'alugar', como argumento dentro do método includes(). Ficando assim:

```
let hasDollarSign = houseRentValue.includes('alugar')
```



Método Includes()



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND
  app.js
  index.html
index.html app.js
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let houseRentValue = 'R$ 1000 - É pra alugar hoje'
4
5 let hasDollarSign = houseRentValue.includes('alugar')
6
```

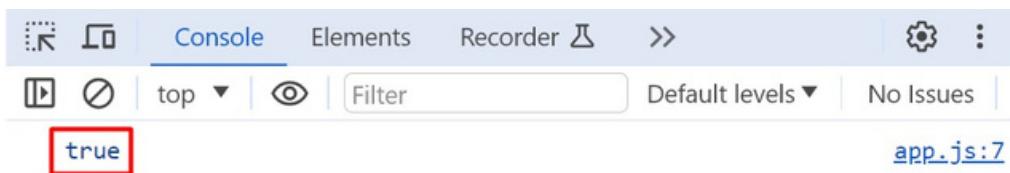
Adicionaremos um console.log passando a hasDollarSign. Ficando assim:

```
console.log(hasDollarSign)
```



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND
  app.js
  index.html
index.html app.js
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let houseRentValue = 'R$ 1000 - É pra alugar hoje'
4
5 let hasDollarSign = houseRentValue.includes('alugar')
6
7 console.log(hasDollarSign)
8
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que true está sendo mostrado.



```
Console Elements Recorder > 
  top ▾ Filter Default levels ▾ No Issues 
  true app.js:7
```



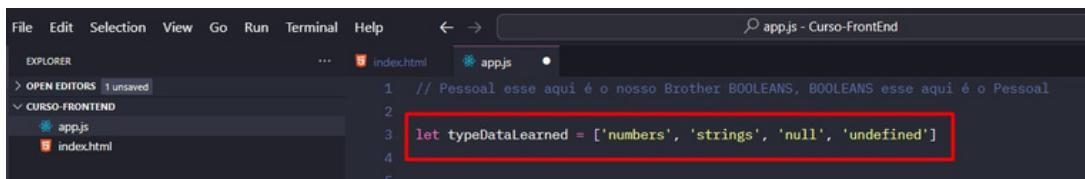
/igor-rebolla

Método Includes()

Agora que já entendemos o conceito de Booleans (false true), vimos exemplos usando o método includes(), tanto para um único caractere quanto para uma string (em ambos os casos retornando tanto false como true).... bora juntar tudo isso, dentro de um array maroto.

Logo abaixo do nosso comentário // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal, daremos um enter maroto e na linha 3, criaremos uma let typeDataLearned que receberá um array [] com os tipos de dados que já aprendemos até aqui (aproveitamos o exemplo aqui para essa pequena revisão) 'numbers', 'strings', 'null', 'undefined'. Ficando assim:

```
let typeDataLearned = ['numbers', 'strings', 'null', 'undefined']
```

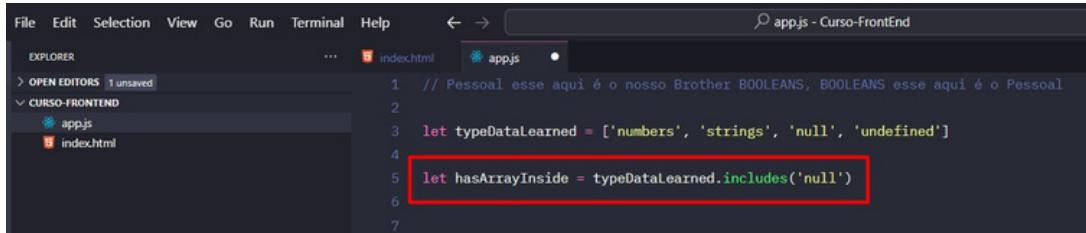


```
// Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
let typeDataLearned = ['numbers', 'strings', 'null', 'undefined']
```

Daremos mais um enter maroto nessa linha 3, para mais uma quebra de linha e na linha 5, declararemos uma nova let hasArrayInside que receberá typeDataLearned.includes() e dentro dos parênteses passaremos a string 'null'. Ficando assim:



Método Includes()



A screenshot of the Visual Studio Code interface. The title bar says "app.js - Curso-FrontEnd". The left sidebar shows an "EXPLORER" view with "OPEN EDITORS" (1 unsaved) and a folder "CURSO-FRONTEND" containing "app.js" and "index.html". The main editor area contains the following code:

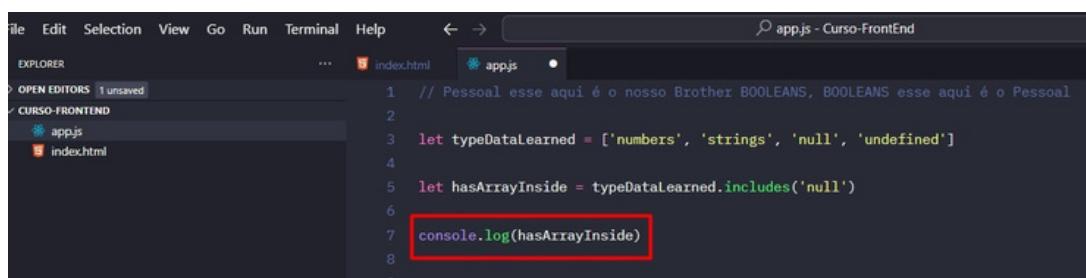
```
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let typeDataLearned = ['numbers', 'strings', 'null', 'undefined']
4
5 let hasArrayInside = typeDataLearned.includes('null')
6
7
```

The line "let hasArrayInside = typeDataLearned.includes('null')"; is highlighted with a red rectangle.

E o nosso nobre método `includes()` verificará se esse valor 'null' consta dentro do array `typeDataLearned`.

Adicionaremos um `console.log` passando a `hasArrayInside`. Ficando assim:

```
console.log(hasArrayInside)
```



A screenshot of the Visual Studio Code interface. The title bar says "app.js - Curso-FrontEnd". The left sidebar shows an "EXPLORER" view with "OPEN EDITORS" (1 unsaved) and a folder "CURSO-FRONTEND" containing "app.js" and "index.html". The main editor area contains the following code:

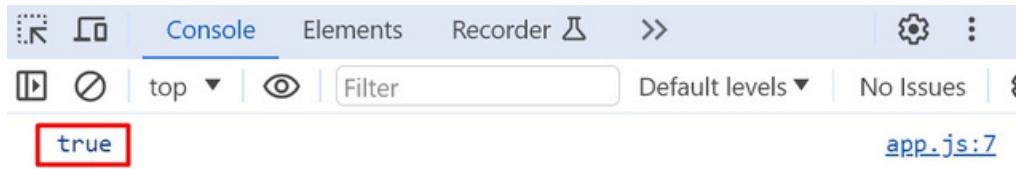
```
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let typeDataLearned = ['numbers', 'strings', 'null', 'undefined']
4
5 let hasArrayInside = typeDataLearned.includes('null')
6
7 console.log(hasArrayInside)
8
```

The line "console.log(hasArrayInside)" is highlighted with a red rectangle.

E ao salvarmos o arquivo `app.js` com aquele `CTRL + S` maroto podemos ver no console do Navegador(Chrome) que true está sendo mostrado.



Método Includes()



Igor, Igor, Igor..... E por que true apareceu ali?

Porque o item 'null' faz parte ou seja está dentro do nosso array typeDataLearned.

E se (assim como quem não quer nada) quisermos alterar o valor para 'object', por exemplo.

Só que antes de vermos esse exemplo, manteremos a nossa let typeDataLearned = ['numbers', 'strings', 'null', 'undefined']. Ficando assim:

A screenshot of the VS Code interface. The Explorer sidebar shows 'OPEN EDITORS 1 unsaved' and 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main editor window shows a snippet of JavaScript code:

```
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let typeDataLearned = ['numbers', 'strings', 'null', 'undefined']
4
5
```

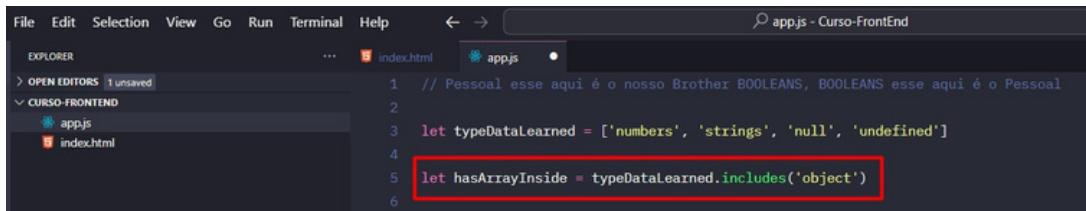
The line 'let typeDataLearned = ['numbers', 'strings', 'null', 'undefined']' is highlighted with a red box.

E na nossa segunda let, passaremos a string 'object', como argumento dentro do método includes(). Ficando assim:

```
let hasArrayInside = typeDataLearned.includes('object')
```



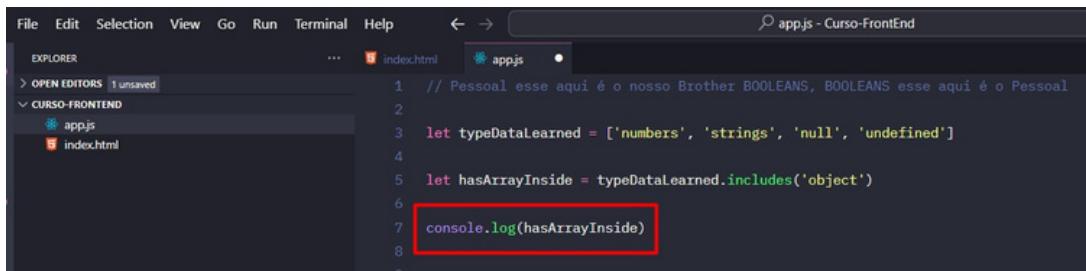
Método Includes()



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let typeDataLearned = ['numbers', 'strings', 'null', 'undefined']
4
5 let hasArrayInside = typeDataLearned.includes('object')
6
```

Adicionaremos um console.log passando a hasArrayInside. Ficando assim:

```
console.log(hasArrayInside)
```



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui é o nosso Brother BOOLEANS, BOOLEANS esse aqui é o Pessoal
2
3 let typeDataLearned = ['numbers', 'strings', 'null', 'undefined']
4
5 let hasArrayInside = typeDataLearned.includes('object')
6
7 console.log(hasArrayInside)
8
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que false está sendo mostrado.



/igor-rebolla

Método Includes()

Igor, Igor, Igor.... E por que isso aconteceu?

Por que o item 'object' não faz parte da turminha ou seja não consta dentro do nosso array typeDataLearned.



Sugestões de prática:

1. Refazer e Entender os exemplos dessa aula;
2. Usar valores definidos por vocês para praticar mais sobre Booleans
3. Usar valores definidos por vocês para praticar mais sobre o Método Includes
4. Notem que nós exemplos vistos nessa aula, utilizamos o let e se tivessemos usados 2 consts ou 1 let e 1 const os resultados teriam sido(s) os mesmos?
Fica mais essa sugestão para vocês realizarem!!!

Colocar isso em prática no VSCode e ver o resultado no console do Navegador.

Programação é prática, curiosidade e repetição!!!!



Operadores de Comparação

Igor, Igor, Igor..... O que são os tais Operadores de Comparação?

Aqui teremos 2 respostas... a primeira resposta será da fonte MDN e a segunda resposta será o que eu entendo sobre Operadores de Comparação.

Fonte MDN Responde essa: Um operador de comparação... compara seus operandos e retorna um valor lógico(Boolean). (FONTE MDN)

Fonte Entendimento do Igor: Pessoal vocês se recordam quando estávamos estudando sobre atribuições a uma variável, nós vimos que igual dentro do JavaScript(JS) não é simbolizado por apenas um sinal de igual (=) e que nós veríamos como o igual seria simbolizado no decorrer do curso.... Eis que esse momento chegou... Bora se apresentarem ai:

```
// Pessoal esse é o duplo sinal de igual (==) que representa igual dentro do Javascript(JS), duplo sinal de igual (==) que representa igual dentro do Javascript(JS) esse aqui é o Pessoal.
```

Agora que nós já sabemos o que são os Operadores de Comparação, veremos alguns exemplos, tanto com Strings quanto com Numbers... pois assim conseguiremos fixar melhor esse conceito.

E bora ver o primeiro exemplo de Operadores de Comparação (Usando Strings).....



Operadores de Comparação

Ah, antes disso..... dentro do nosso nobre arquivo app.js, na linha 1 criaremos um comentário: // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal,

Então logo abaixo do comentário: // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal, declararemos uma let firstCourseName que recebe 'front'. Ficando assim:

```
let firstCourseName = 'front'
```



```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let firstCourseName = 'front'
4
```

Adicionaremos um console.log passando a firstCourseName que é igual (==) a 'front'. Ficando assim:

```
console.log(firstCourseName == 'front')
```



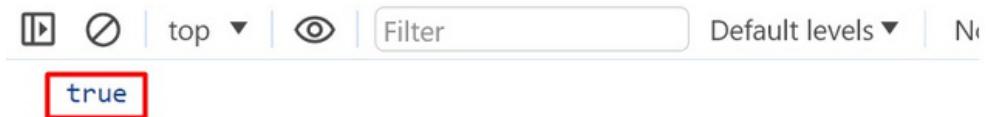
```
File Edit Selection View Go Run Terminal Help ← → 🔍 app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 3 unsaved ...
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let firstCourseName = 'front'
4
5 console.log(firstCourseName == 'front')
6
```



/igor-rebolla

Operadores de Comparação

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que true está sendo mostrado no console.



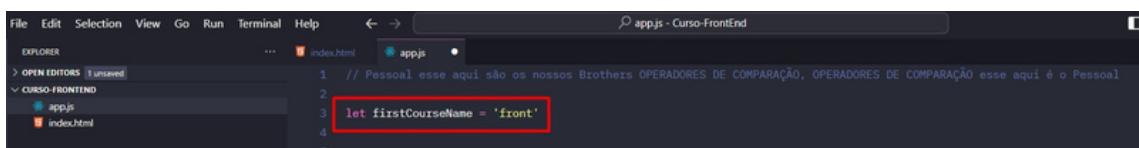
E porque True apareceu no exemplo acima?

Porque o duplo sinal de igual (==) fez uma verificação marota se o valor da let firstCourseName é igual == 'front', como o valor da expressão firstCourseName == 'front' foi comparado que sim, então foi retornado true.

Operadores de Comparaçāo

No nosso segundo exemplo de Operadores de Comparaçāo (Strings), manteremos sem nenhuma alteraçāo a nossa let firstCourseName = 'front' . Ficando assim:

```
let firstCourseName = 'front'
```



```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
index.html
app.js
1 // Pessoal esse aqui sāo os nossos Brothers OPERADORES DE COMPARAÇĀO, OPERADORES DE COMPARAÇĀO esse aqui é o Pessoal
2
3 let firstCourseName = 'front'
4
```

Adicionaremos um console.log passando a firstCourseName que é igual (==) a 'Front' só que com o 'F' em maiúsculo. Ficando assim:

```
console.log(firstCourseName == 'Front')
```



```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
index.html
app.js
1 // Pessoal esse aqui sāo os nossos Brothers OPERADORES DE COMPARAÇĀO, OPERADORES DE COMPARAÇĀO esse aqui é o Pessoal
2
3 let firstCourseName = 'front'
4
5 console.log(firstCourseName == 'Front')
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que false estā sendo mostrado no console.



Operadores de Comparação



Igor, Igor, Igor, Igor, Igor.... Socorre a gente aqui, please.... Por que retornou false, sendo que as palavras são iguais?

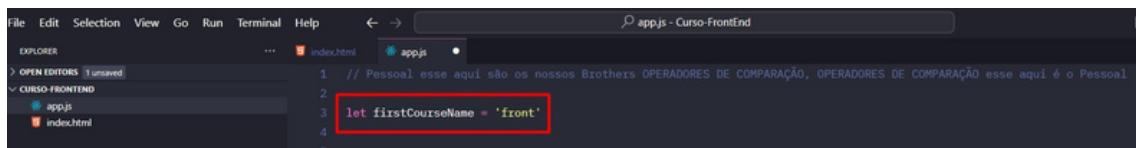
Porque essa string que está em `let firstCourseName = 'front'` , é diferente da string 'Front' que está dentro do nosso `console.log`, e no nosso exemplo como a primeira letra de uma das palavras é maiúscula no caso a letra 'F' dentro do `console.log`, e essa alteração mínima é mais do que o suficiente para que o JavaScript(JS) faça com que essas strings sejam diferentes.



Operadores de Comparação

No nosso terceiro exemplo de Operadores de Comparação (Strings), manteremos sem nenhuma alteração a nossa let firstCourseName = 'front' . Ficando assim:

```
let firstCourseName = 'front'
```



```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 3 unsaved CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let firstCourseName = 'front'
4
```

Adicionaremos um console.log passando a firstCourseName que será maior > que 'end' com o 'e' em minúsculo. Ficando assim:

```
console.log(firstCourseName > 'end')
```



```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 3 unsaved CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let firstCourseName = 'front'
4
5 console.log(firstCourseName > 'end')
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que true está sendo mostrado no console.



Operadores de Comparação



Igor, Igor, Igor, Igor, Igor..... eita que o negócio não entrou na nossa cabeça, como essa comparação é feita usando o sinal de maior > que. O que aconteceu aqui entre a string 'front' e a string 'end'. Pode esclarecer isso pra gente?

Claro, só ser for agora... peço a gentileza que isso aqui seja bem compreendido, pois aqui tem um detalhe que vai fazer a diferença.

O detalhe é o seguinte: Aqui no nosso exemplo ele está nos dizendo que 'front' é maior que 'end'... pois bem: no nosso alfabeto essa primeira letra(f) de 'front' da let firstCourseName = 'front' ela vem depois da letra 'e' de 'end'... então no nosso exemplo 'front' é maior que 'end'. Podemos entender com base nisso que as últimas letras do alfabeto são maiores que as primeiras letras do alfabeto.

O que esse detalhe acima quer dizer: Vamos pegar as 26 letras do alfabeto e colocaremos aqui embaixo (todas em minúsculas).

a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z.

Onde:



Operadores de Comparação

a - Primeiro Valor
b - Segundo Valor
c - Terceiro Valor
d - Quarto Valor
e - Quinto Valor
f - Sexto Valor
g - Sétimo Valor
h - Oitavo Valor
i - Nono Valor
j - Décimo Valor
k - Décimo Primeiro Valor
l - Décimo Segundo Valor
m - Décimo Terceiro Valor
n - Décimo Quarto Valor
o - Décimo Quinto Valor
p - Décimo Sexto Valor
q - Décimo Sétimo Valor
r - Décimo Oitavo Valor
s - Décimo Nono Valor
t - Vigésimo Valor
u - Vigésimo Primeiro Valor
v - Vigésimo Segundo Valor
w - Vigésimo Terceiro Valor
x - Vigésimo Quarto Valor
y - Vigésimo Quinto Valor
z - Vigésimo Sexto Valor



Operadores de Comparação

Agora nós sabemos que as últimas letras do nosso alfabeto valem mais do que as primeiras. Chegamos a conclusão que:

a letra (string) f - é o nosso Sexto Valor

a letra (string) e - é o nosso Quinto Valor

f - é o nosso Sexto Valor > e - é o nosso Quinto Valor

Logo a string 'f' que faz parte da string 'front' é maior > que a string 'e' que faz parte da string 'end'.



Operadores de Comparação

No nosso quarto exemplo de Operadores de Comparação (Strings), manteremos sem nenhuma alteração a nossa let firstCourseName = 'front' . Ficando assim:

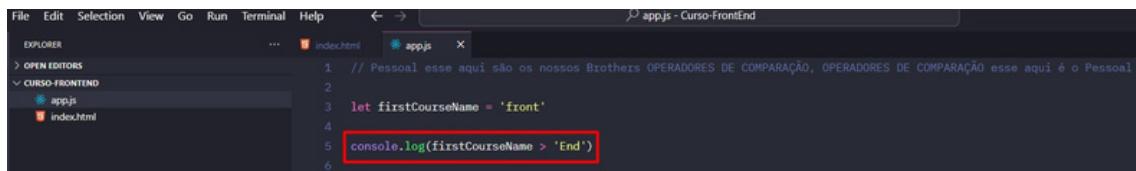
```
let firstCourseName = 'front'
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND app.js index.html
app.js
index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let firstCourseName = 'front'
4
5
```

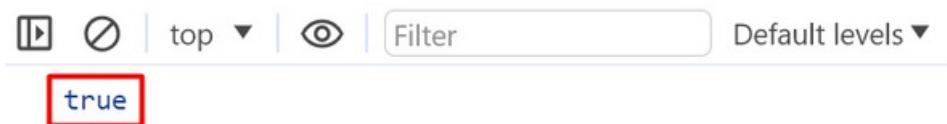
Adicionaremos um console.log passando a firstCourseName que será maior > que 'End' com o 'E' em maiúsculo. Ficando assim:

```
console.log(firstCourseName > 'End')
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS ...
CURSO-FRONTEND app.js index.html
app.js
index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let firstCourseName = 'front'
4
5 console.log(firstCourseName > 'End')
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que true está sendo mostrado no console.



```
▶ ⏷ top ⏵ Filter Default levels ▾
true
```



Operadores de Comparação

Igor, Igor, Igor, Igor, Igor..... eita que o negócio não entrou na nossa cabeça (parte 2), nós entendemos como funciona comparação entre 'strings' com minúsculas. O que aconteceu entre a string 'front' e a string 'End' agora com o 'E' em maiúsculo. Pode esclarecer isso pra gente?

Claro, só ser for agora... peço a gentileza que isso aqui também seja bem compreendido, pois aqui tem um novo detalhe que vai fazer a diferença.

E por que o TRUE acima foi exibido? Por que nesse exemplo não somente a letra 'f' minúscula da string 'front' na let firstCourseName é maior que a letra 'E' maiúscula, ela a letra 'f' minúscula é maior do que qualquer letra maiúscula, no caso aqui o (E) de 'End'.

O que esse detalhe acima quer dizer: Pegaremos agora 2 alfabetos com as 26 letras do alfabeto (cada) e colocaremos aqui embaixo (o primeiro alfabeto com as letras em minúsculas e o segundo alfabeto com as letras em maiúsculas).

Primeiro alfabeto (Letras minúsculas)

a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z.

Segundo alfabeto (Letras maiúsculas)

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.

Onde



Operadores de Comparação

a - Primeiro Valor
b - Segundo Valor
c - Terceiro Valor
d - Quarto Valor
e - Quinto Valor
f - Sexto Valor
g - Sétimo Valor
h - Oitavo Valor
i - Nono Valor
j - Décimo Valor
k - Décimo Primeiro Valor
l - Décimo Segundo Valor
m - Décimo Terceiro Valor
n - Décimo Quarto Valor
o - Décimo Quinto Valor
p - Décimo Sexto Valor
q - Décimo Sétimo Valor
r - Décimo Oitavo Valor
s - Décimo Nono Valor
t - Vigésimo Valor
u - Vigésimo Primeiro Valor
v - Vigésimo Segundo Valor
w - Vigésimo Terceiro Valor
x - Vigésimo Quarto Valor
y - Vigésimo Quinto Valor
z - Vigésimo Sexto Valor

A - Primeiro Valor
B - Segundo Valor
C - Terceiro Valor
D - Quarto Valor
E - Quinto Valor
F - Sexto Valor
G - Sétimo Valor
H - Oitavo Valor
I - Nono Valor
J - Décimo Valor
K - Décimo Primeiro Valor
L - Décimo Segundo Valor
M - Décimo Terceiro Valor
N - Décimo Quarto Valor
O - Décimo Quinto Valor
P - Décimo Sexto Valor
Q - Décimo Sétimo Valor
R - Décimo Oitavo Valor
S - Décimo Nono Valor
T - Vigésimo Valor
U - Vigésimo Primeiro Valor
V - Vigésimo Segundo Valor
W - Vigésimo Terceiro Valor
X - Vigésimo Quarto Valor
Y - Vigésimo Quinto Valor
Z - Vigésimo Sexto Valor



/igor-rebolla

Operadores de Comparação

Notem que os valores tanto do alfabeto das letras minúsculas quanto do alfabeto das letras maiúsculas, são idênticos... só que para o JavaScript(JS)... qualquer letra minúscula independente de qual seja, será sempre maior do que qualquer letra maiúscula.

Chegamos a conclusão que:

a letra (string) f - por ser minúscula vai ser maior > que
a letra (string) E - por ser maiúscula independente do valor que ela tenha

f - por ser minúscula vai ser maior > que E - por ser maiúscula

Birutão isso né? Só um ‘cadinho’... agora no nosso próximo exemplo misturaremos tudo pra aprendermos juntos e assim fixarmos melhor esse assunto.

Bora ver isso na próxima página.....



Operadores de Comparação

No nosso quinto exemplo de Operadores de Comparação (Strings), manteremos sem nenhuma alteração a nossa let firstCourseName = 'front' . Ficando assim:

```
let firstCourseName = 'front'
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let firstCourseName = 'front'
4
5
```

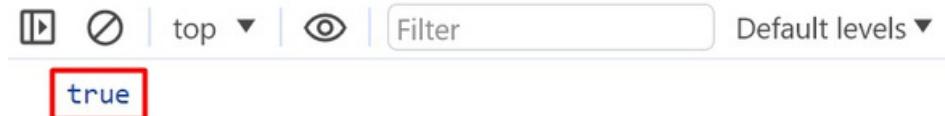
Adicionaremos um console.log passando a firstCourseName que será maior > que 'Front' com o 'F' em maiúsculo. Ficando assim:

```
console.log(firstCourseName > 'Front')
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let firstCourseName = 'front'
4
5 console.log(firstCourseName > 'Front')
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que true está sendo mostrado no console.



```
top Filter Default levels ▾
true
```



/igor-rebolla

Operadores de Comparação

E porque true apareceu no exemplo acima?

Porque como vimos que qualquer letra minúscula será sempre maior que qualquer letra maiúscula.

Pois bem, aqui temos uma letra minúscula 'f' será sempre maior > que uma letra maiúscula 'F'. Por isso true foi retornado.

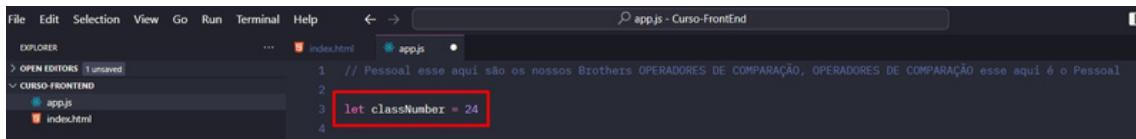


Operadores de Comparação

Agora que nós vimos as comparações entre STRINGS, chegou o grande momento de vermos comparações com NÚMEROS.

Então bora ver o primeiro exemplo sobre Números.... Yuppppp!!!!

Como nós já definimos na linha 1 do arquivo app.js o nosso comentário // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal, daremos um enter maroto e na linha 3, criaremos uma let classNumber que receberá 24. Ficando assim:



The screenshot shows the VS Code interface with the 'app.js' file open. The code contains a multi-line comment at the top and a single line of code 'let classNumber = 24' on line 3. The line 'let classNumber = 24' is highlighted with a red rectangle.

```
// Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
let classNumber = 24
```

Adicionaremos um console.log passando a classNumber que é igual (==) utilizando dois sinais de igual a 24. Ficando assim:

```
console.log(classNumber == 24)
```



The screenshot shows the VS Code interface with the 'app.js' file open. The code now includes a 'console.log' statement on line 5, which is highlighted with a red rectangle. The rest of the code remains the same as in the previous screenshot.

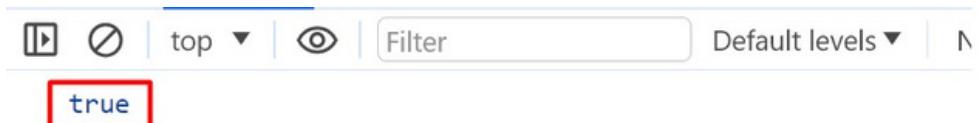
```
// Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
let classNumber = 24
console.log(classNumber == 24)
```



/igor-rebolla

Operadores de Comparação

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que true está sendo mostrado no console.



E porque true apareceu no exemplo acima?

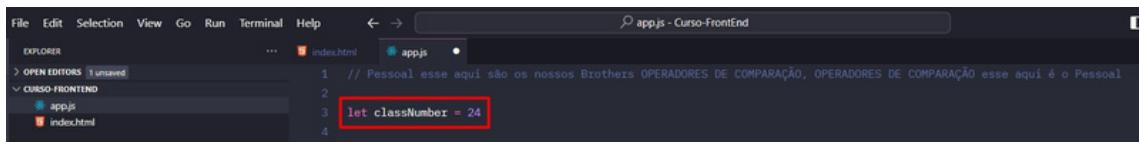
Porque assim como vimos com o primeiro exemplo dessa aula em strings, o duplo sinal de igual (==) fez uma verificação marota se o valor da let classNumber é igual == a 24, se for igual..... essa expressão classNumber == 24 retornará true.



Operadores de Comparação

No nosso segundo exemplo de Operadores de Comparação (Numbers), manteremos sem nenhuma alteração a nossa let classNumber = 24 . Ficando assim:

```
let classNumber = 24
```



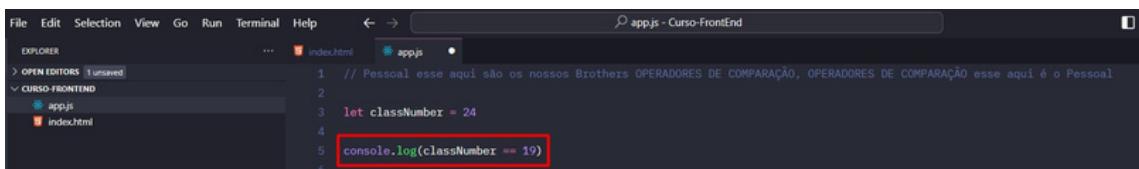
A screenshot of a code editor window titled "app.js - Curso-FrontEnd". The file contains the following code:

```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
```

The line "let classNumber = 24" is highlighted with a red rectangle.

Adicionaremos um console.log passando a classNumber que é igual == a 19. Ficando assim:

```
console.log(classNumber == 19)
```



A screenshot of a code editor window titled "app.js - Curso-FrontEnd". The file contains the following code:

```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
  app.js
  index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
5 console.log(classNumber == 19)
6
```

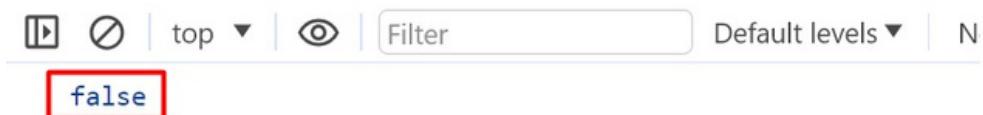
The line "console.log(classNumber == 19)" is highlighted with a red rectangle.



/igor-rebolla

Operadores de Comparação

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que false está sendo mostrado no console.



E porque false apareceu no exemplo acima?

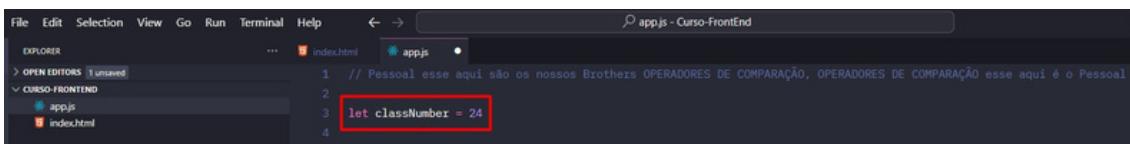
O nosso amigão duplo sinal de igual (==) fez uma verificação marota se o valor da let classNumber é igual a 24, se for igual como o valor da expressão classNumber == 19..... e aqui foi comparado que não, então foi retornado false.



Operadores de Comparação

No nosso terceiro exemplo de Operadores de Comparação (Numbers), manteremos sem nenhuma alteração a nossa `let classNumber = 24`. Ficando assim:

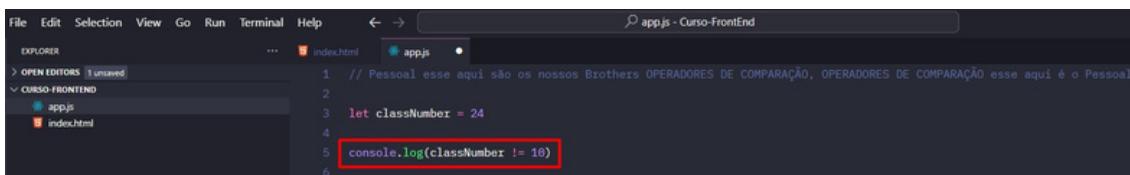
```
let classNumber = 24
```



```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved
COURSE-FRONTEND app.js index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
```

Adicionaremos um `console.log` passando a `classNumber` que não será igual, representada pelo comparador `!=` seguido do número 10. Ficando assim:

```
console.log(classNumber != 10)
```

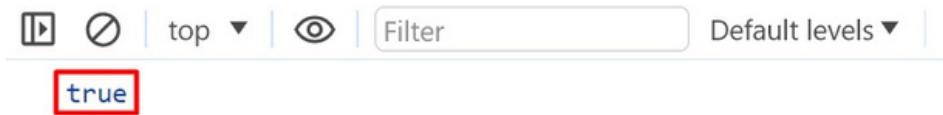


```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved
COURSE-FRONTEND app.js index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
5 console.log(classNumber != 10)
6
```

E ao salvarmos o arquivo `app.js` com aquele `CTRL + S` maroto podemos ver no `console` do Navegador(Chrome) que `true` está sendo mostrado no `console`.



Operadores de Comparação



E porque true apareceu no exemplo acima?

Porque o jeito certinho para efetuarmos a leitura da expressão que está dentro do nosso console.log é: classNumber não é igual a 10.

Depois que fizemos essa leitura, é perguntado:

Se isso é verdade(true)?

E a resposta recebida, é de que sim, isso é verdade(true).

É feita outra pergunta:

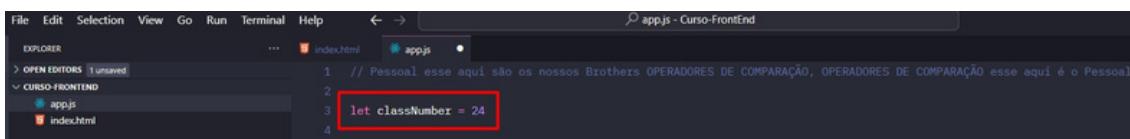
E porque isso é verdade(true)?

Porque 24 é não é igual a 10 ou seja: 24 != 10

Operadores de Comparação

No nosso quarto exemplo de Operadores de Comparação (Numbers), manteremos sem nenhuma alteração a nossa let classNumber = 24 . Ficando assim:

```
let classNumber = 24
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
```

Adicionaremos um console.log passando a classNumber que não será igual, representada pelo comparador != seguido do número 24. Ficando assim:

```
console.log(classNumber != 24)
```

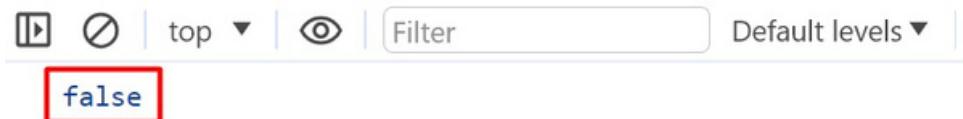


```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
5 console.log(classNumber != 24)
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que false está sendo mostrado no console.



Operadores de Comparação



E porque false apareceu no exemplo acima?

Porque o jeito certinho para efetuarmos a leitura da expressão que está dentro do nosso console.log é: classNumber não é igual a 24.

Depois que fizemos essa leitura, é perguntado:

Se isso é verdade (true)?

E a resposta recebida, é de que não, isso é falso(false).

É feita outra pergunta:

E porque isso é falso(false)?

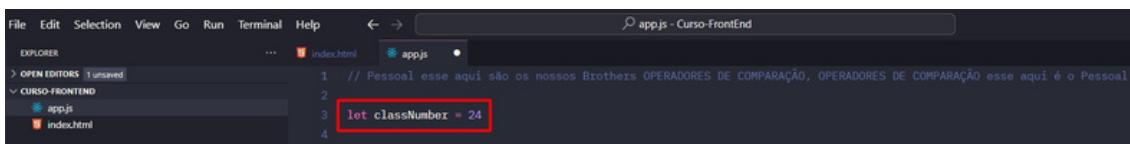
Porque 24 é igual a 24 ou seja: 24 == 24



Operadores de Comparação

No nosso quinto exemplo de Operadores de Comparação (Numbers), manteremos sem nenhuma alteração a nossa `let classNumber = 24`. Ficando assim:

```
let classNumber = 24
```



```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
```

Adicionaremos um `console.log` passando a `classNumber` que é maior `>` que `3`, representada pelo comparador `>` seguido do número `3`. Ficando assim:

```
console.log(classNumber > 3)
```



```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND app.js index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
5 console.log(classNumber > 3)
6
```

E ao salvarmos o arquivo `app.js` com aquele `CTRL + S` maroto podemos ver no `console` do Navegador(Chrome) que `true` está sendo mostrado no `console`.



Operadores de Comparação



E porque true apareceu no exemplo acima?

Porque o jeito certinho para efetuarmos a leitura da expressão que está dentro do nosso console.log é: classNumber é maior > que 3.

Depois que fizemos essa leitura, é perguntado:

Se isso é verdade (true)?

E a resposta recebida, é de sim, isso é verdade(true).

É feita outra pergunta:

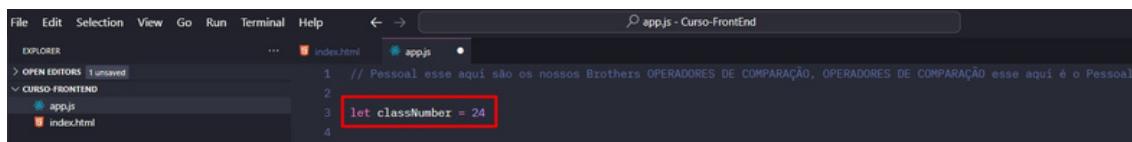
E porque isso é verdade(true)?

Porque 24 é maior que 3 ou seja: 24 > 3

Operadores de Comparaçāo

No nosso sexto exemplo de Operadores de Comparaçāo (Numbers), manteremos sem nenhuma alteraçāo a nossa let classNumber = 24 . Ficando assim:

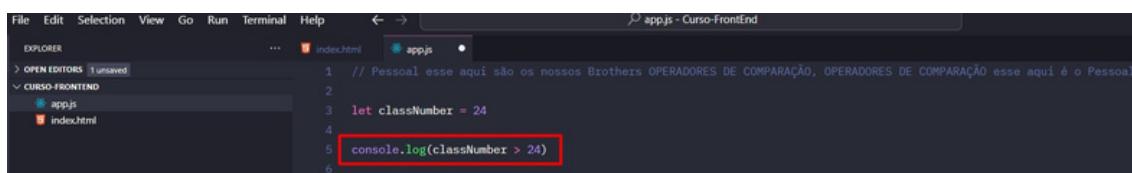
```
let classNumber = 24
```



```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui sāo os nossos Brothers OPERADORES DE COMPARAÇĀO, OPERADORES DE COMPARAÇĀO esse aqui é o Pessoal
2
3 let classNumber = 24
4
```

Adicionaremos um console.log passando a classNumber que é maior > que 24, representada pelo comparador > seguido do n mero 24. Ficando assim:

```
console.log(classNumber > 24)
```

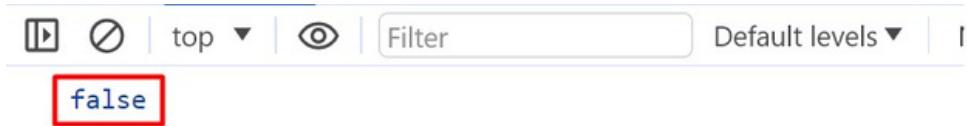


```
File Edit Selection View Go Run Terminal Help ← → app.js - Curso-FrontEnd
EXPLORER ... index.html app.js
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui s o os nossos Brothers OPERADORES DE COMPARA O, OPERADORES DE COMPARA O esse aqui ´ o Pessoal
2
3 let classNumber = 24
4
5 console.log(classNumber > 24)
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que false est a sendo mostrado no console.



Operadores de Comparação



E porque false apareceu no exemplo acima?

Porque o jeito certinho para efetuarmos a leitura da expressão que está dentro do nosso console.log é: classNumber é maior > que 24.

Depois que fizemos essa leitura, é perguntado:

Se isso é verdade (true)?

E a resposta recebida, é de que não, isso é falso(false).

É feita outra pergunta:

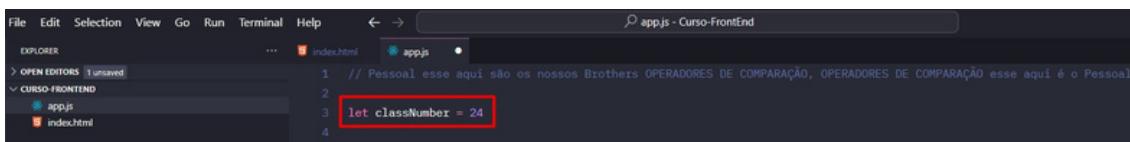
E porque isso é falso(false)?

Porque 24 não é maior que 24, e sim igual a 24 ou seja: $24 == 24$

Operadores de Comparação

No nosso sétimo exemplo de Operadores de Comparação (Numbers), manteremos sem nenhuma alteração a nossa `let classNumber = 24`. Ficando assim:

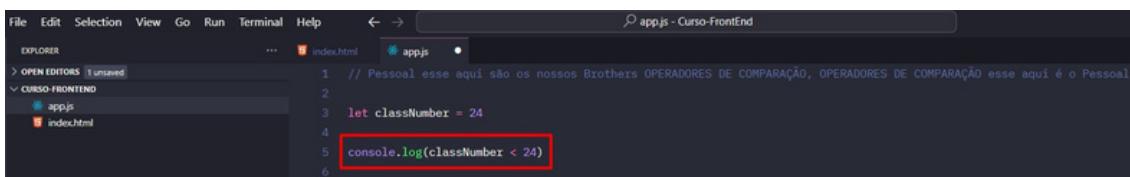
```
let classNumber = 24
```



```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
```

Adicionaremos um `console.log` passando a `classNumber` que é menor `<` que 24, representada pelo comparador `<` seguido do número 24. Ficando assim:

```
console.log(classNumber < 24)
```

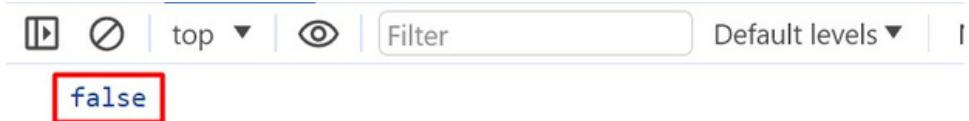


```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 1 unsaved
CURSO-FRONTEND
app.js
index.html
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
5 console.log(classNumber < 24)
6
```

E ao salvarmos o arquivo `app.js` com aquele `CTRL + S` maroto podemos ver no `console` do Navegador(Chrome) que `false` está sendo mostrado no `console`.



Operadores de Comparação



E porque false apareceu no exemplo acima?

Porque o jeito certinho para efetuarmos a leitura da expressão que está dentro do nosso console.log é: classNumber é menor < que 24.

Depois que fizemos essa leitura, é perguntado:

Se isso é verdade (true)?

E a resposta recebida, é de que não, isso é falso(false).

É feita outra pergunta:

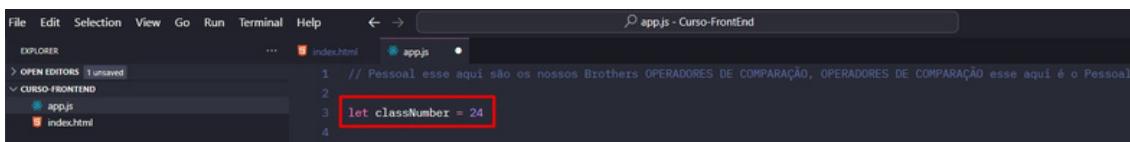
E porque isso é falso(false)?

Porque 24 não é menor que 24, e sim igual a 24 ou seja: 24 == 24

Operadores de Comparação

No nosso oitavo exemplo de Operadores de Comparação (Numbers), manteremos sem nenhuma alteração a nossa let classNumber = 24 . Ficando assim:

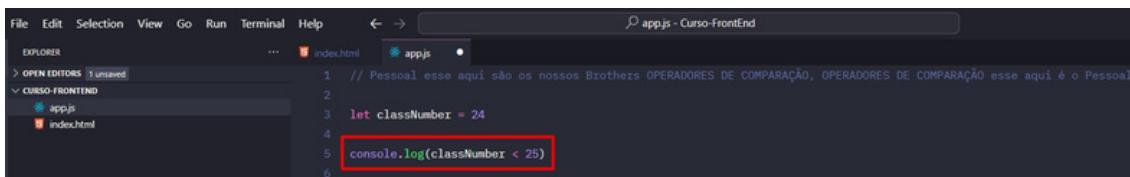
```
let classNumber = 24
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND app.js index.html
index.html app.js
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
```

Adicionaremos um console.log passando a classNumber que é menor < que 25, representada pelo comparador < seguido do número 25. Ficando assim:

```
console.log(classNumber < 25)
```

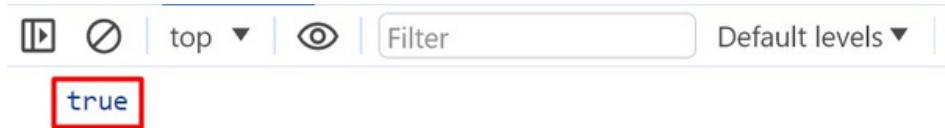


```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS 1 unsaved ...
CURSO-FRONTEND app.js index.html
index.html app.js
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
5 console.log(classNumber < 25)
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que false está sendo mostrado no console.



Operadores de Comparação



E porque true apareceu no exemplo acima?

Porque o jeito certinho para efetuarmos a leitura da expressão que está dentro do nosso console.log é: classNumber é menor < que 25.

Depois que fizemos essa leitura, é perguntado:

Se isso é verdade (true)?

E a resposta recebida, é de que sim, isso é verdade(true).

É feita outra pergunta:

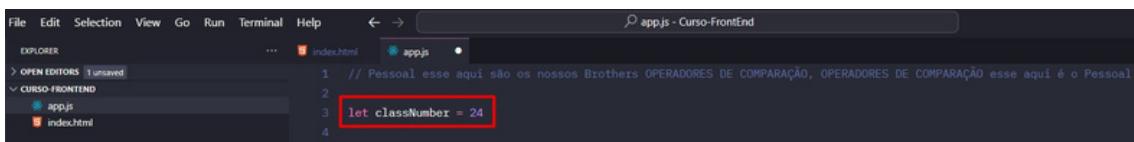
E porque isso é verdade(true)?

Porque 24 é menor que 25 ou seja: $24 < 25$

Operadores de Comparação

No nosso nono exemplo de Operadores de Comparação (Numbers), manteremos sem nenhuma alteração a nossa `let classNumber = 24`. Ficando assim:

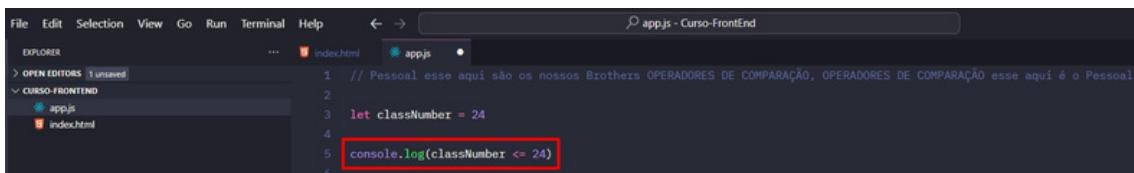
```
let classNumber = 24
```



A screenshot of the Visual Studio Code interface. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help. The title bar says 'app.js - Curso-FrontEnd'. The Explorer sidebar shows 'OPEN EDITORS 1 unsaved' and 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main editor area has four lines of code: 1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal, 2, 3 let classNumber = 24, 4. The line 'let classNumber = 24' is highlighted with a red rectangle.

Adicionaremos um `console.log` passando a `classNumber` que é menor ou igual (`<=`) que 24, representada pelos comparadores `<=` seguido do número 24. Ficando assim:

```
console.log(classNumber <= 24)
```

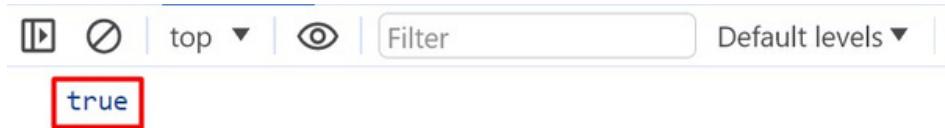


A screenshot of the Visual Studio Code interface. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help. The title bar says 'app.js - Curso-FrontEnd'. The Explorer sidebar shows 'OPEN EDITORS 1 unsaved' and 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main editor area has six lines of code: 1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal, 2, 3 let classNumber = 24, 4, 5 console.log(classNumber <= 24), 6. The line 'console.log(classNumber <= 24)' is highlighted with a red rectangle.

E ao salvarmos o arquivo `app.js` com aquele `CTRL + S` maroto podemos ver no console do Navegador(Chrome) que true está sendo mostrado no console.



Operadores de Comparação



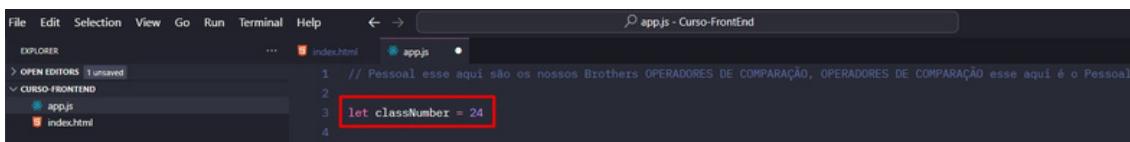
E porque true apareceu no exemplo acima?

Porque ao inserirmos que a classNumber é menor ou igual (\leq) a 24. Então ao adicionarmos um sinal de = logo após esse sinal de menor que, esse sinal vira um menor ou igual que. E esse OU marotinho aqui faz uma diferença enorme, porque 24 não é menor que 24, mas como esse sinal \leq verifica se o 24 é menor ou igual a 24, essa expressão classNumber \leq 24 é verdadeira(true).

Operadores de Comparação

No nosso décimo exemplo de Operadores de Comparação (Numbers), manteremos sem nenhuma alteração a nossa let classNumber = 24 . Ficando assim:

```
let classNumber = 24
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS CURSO-FRONTEND index.html app.js
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
```

Adicionaremos um console.log passando a classNumber que é maior ou igual (\geq) que 24, representada pelos comparadores \geq seguido do número 24. Ficando assim:

```
console.log(classNumber  $\geq$  24)
```

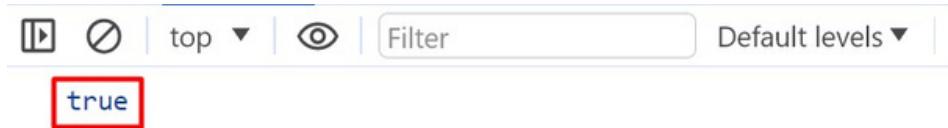


```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS CURSO-FRONTEND index.html app.js
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
5 console.log(classNumber  $\geq$  24)
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que true está sendo mostrado no console.



Operadores de Comparação



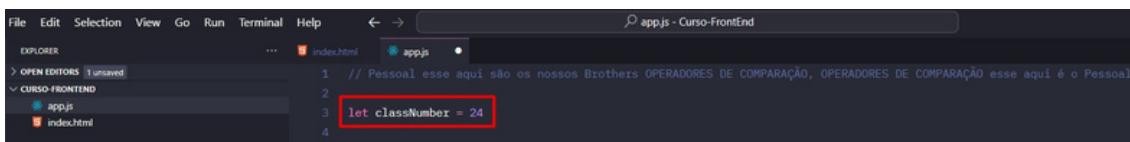
E porque true apareceu no exemplo acima?

Porque ao inserirmos que a classNumber é maior ou igual (\geq) a 24. Então ao adicionarmos um sinal de = logo após esse sinal de maior que, esse sinal vira um maior ou igual que. E esse OU marotinho faz uma diferença enorme, porque 24 não é maior que 24, mas como esse sinal \geq verifica se o 24 é maior ou igual a 24, essa expressão `classNumber \geq 24` é verdadeira.

Operadores de Comparação

No nosso décimo primeiro exemplo de Operadores de Comparação (Numbers), manteremos sem nenhuma alteração a nossa let classNumber = 24 . Ficando assim:

```
let classNumber = 24
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS CURSO-FRONTEND index.html app.js
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
```

Adicionaremos um console.log passando a classNumber que é maior ou igual (\geq) que 20, representada pelos comparadores \geq seguido do número 20. Ficando assim:

```
console.log(classNumber  $\geq$  20)
```

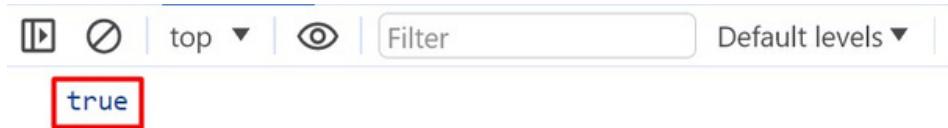


```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS CURSO-FRONTEND index.html app.js
1 // Pessoal esse aqui são os nossos Brothers OPERADORES DE COMPARAÇÃO, OPERADORES DE COMPARAÇÃO esse aqui é o Pessoal
2
3 let classNumber = 24
4
5 console.log(classNumber  $\geq$  20)
6
```

E ao salvarmos o arquivo app.js com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que true está sendo mostrado no console.



Operadores de Comparação



E porque true apareceu no exemplo acima?

Essa resposta eu vou deixar para vocês pesquisarem.

Dica: tomem como base os últimos 2 exemplos (Lembrem do OU marotinho)



Sugestões de prática:

1. Refazer e Entender os exemplos dessa aula;
2. Usar valores definidos por vocês para praticar mais sobre Operadores de Comparação com Strings;
3. Usar valores definidos por vocês para praticar mais sobre Operadores de Comparação com Numbers;
4. Notem que nós exemplos vistos nessa aula, utilizamos o let e se tivéssemos utilizado a const os resultados teriam sido(s) os mesmos?
Fica mais essa sugestão para vocês realizarem!!!

Colocar isso em prática no VSCode e ver o resultado no console do Navegador.

Programação é prática, curiosidade e repetição!!!



Pergunta do Amigo Internauta: Igor de São Paulo - SP

Na páginas 9 e 13 dessa aula nós vimos exemplos com o alfabeto começando com o caractere 'A' como primeiro valor, o caractere 'B' como segundo valor..... o caractere 'Z' como vigésimo sexto valor (tanto para o alfabeto na página 9 em minúsculo quanto para os 2 alfabetos - página 13 em minúsculo e maiúsculo).

Lembram da nossa aula 19 quando vimos que o JavaScript(JS) é zero-based(baseado em zero)..... Faria sentido o primeiro caractere tanto do alfabeto da página 9, quanto os alfabetos da pagina 13 começarem com o valor 0 ao invés de 1 (primeiro valor)?

