

Curso: Front-End do Bem



AULA 19

- Numbers (Parte Final)
- Exemplos de Numbers (Parte Final)
 - Strings
 - Exemplos de Strings
- Sugestões de Prática para essa aula
- O que vamos aprender na aula 20

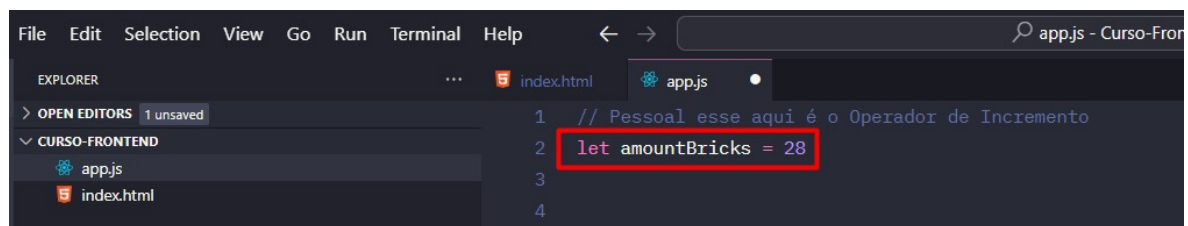
Numbers - Parte Final

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
```

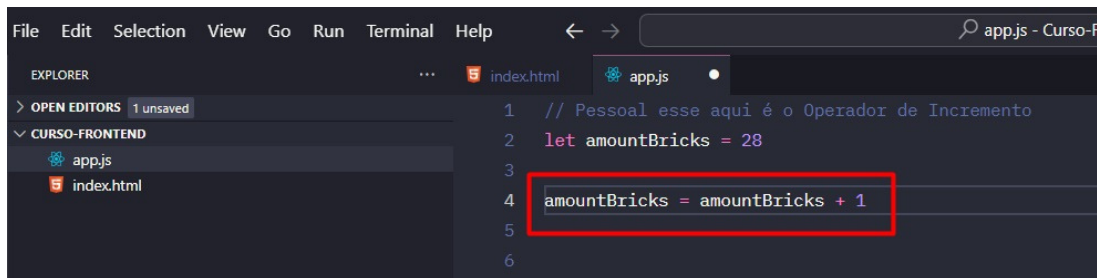


Imagina que isso é o número de tijolos (amountBricks) que uma parede de uma casa tem e se quisermos adicionar + 1 tijolo para essa let. Podemos fazer desse jeito, especificaremos que amountBricks recebe amountBricks + 1 (amountBricks = amountBricks + 1) ou seja amountBricks recebe o valor que ela já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

```
amountBricks = amountBricks + 1
```



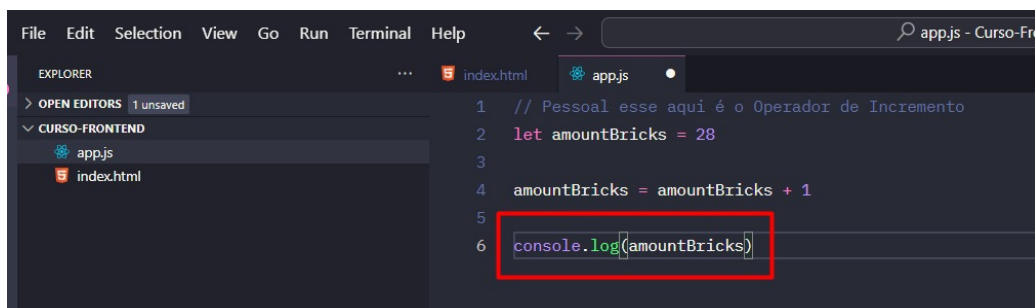
Numbers - Parte Final



```
1 // Pessoal esse aqui é o Operador de Incremento
2 let amountBricks = 28
3
4 amountBricks = amountBricks + 1
5
6
```

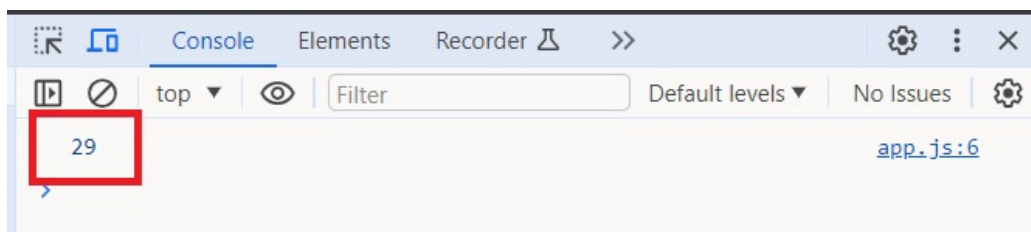
Adicionaremos um `console.log` passando a `amountBricks`. Ficando assim:

`console.log(amountBricks)`



```
1 // Pessoal esse aqui é o Operador de Incremento
2 let amountBricks = 28
3
4 amountBricks = amountBricks + 1
5
6 console.log(amountBricks)
```

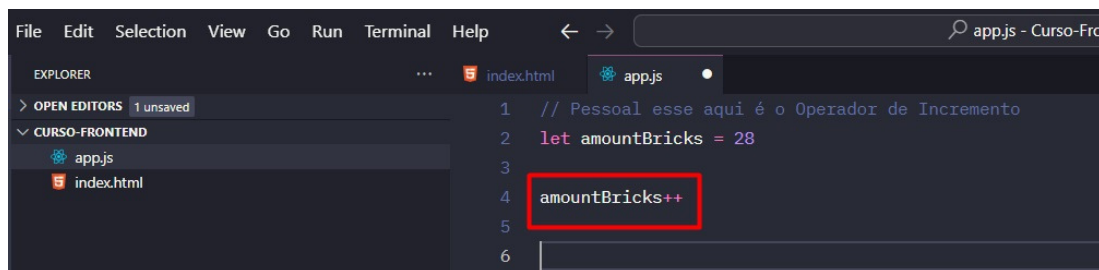
E ao salvarmos o arquivo `app.js` com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o número 29 é exibido.



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++`



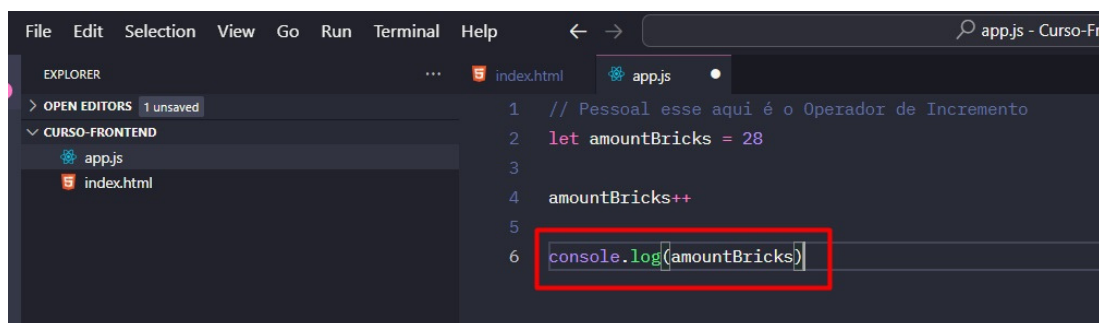
A screenshot of the Visual Studio Code editor interface. The Explorer panel on the left shows a project named 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main editor window displays the 'app.js' file with the following code:

```
1 // Pessoal esse aqui é o Operador de Incremento
2 let amountBricks = 28
3
4 amountBricks++
5
6
```

The line `amountBricks++` on line 4 is highlighted with a red rectangular box.

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

`console.log(amountBricks)`



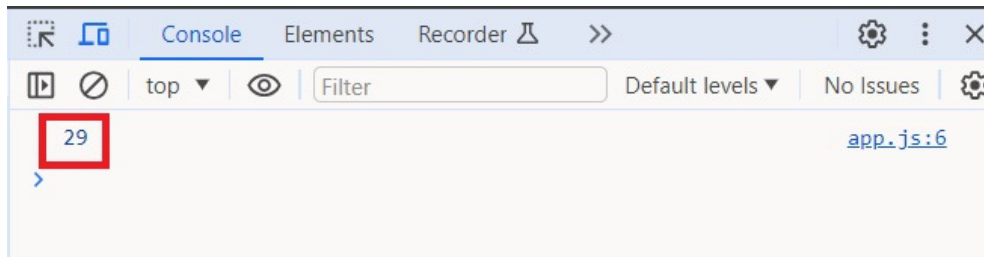
A screenshot of the Visual Studio Code editor interface, similar to the previous one. The main editor window displays the 'app.js' file with the following code:

```
1 // Pessoal esse aqui é o Operador de Incremento
2 let amountBricks = 28
3
4 amountBricks++
5
6 console.log(amountBricks)
```

The line `console.log(amountBricks)` on line 6 is highlighted with a red rectangular box.

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.



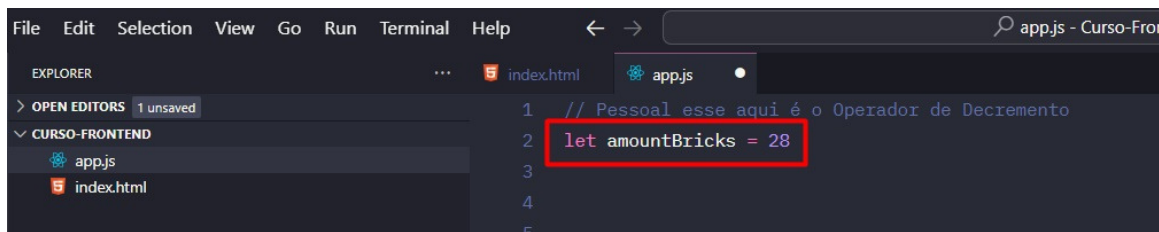
O nome do que acabamos de fazer agora é o incremento, nós incrementamos em 1 o valor da variável ou seja fizemos com que ela recebesse todo o valor que ela já tinha + 1.

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:

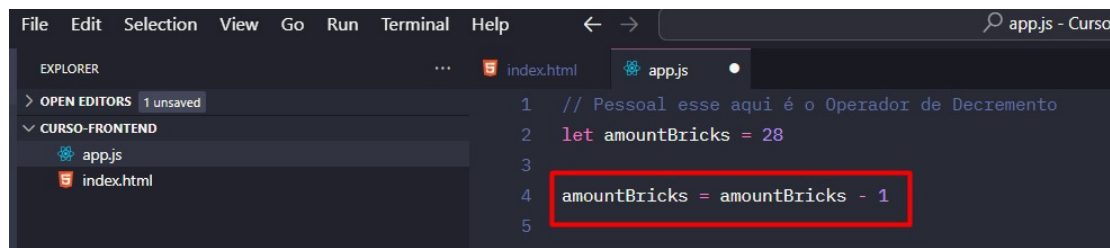
```
// Pessoal esse aqui é o Operador de Decremento  
let amountBricks = 28
```



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
```

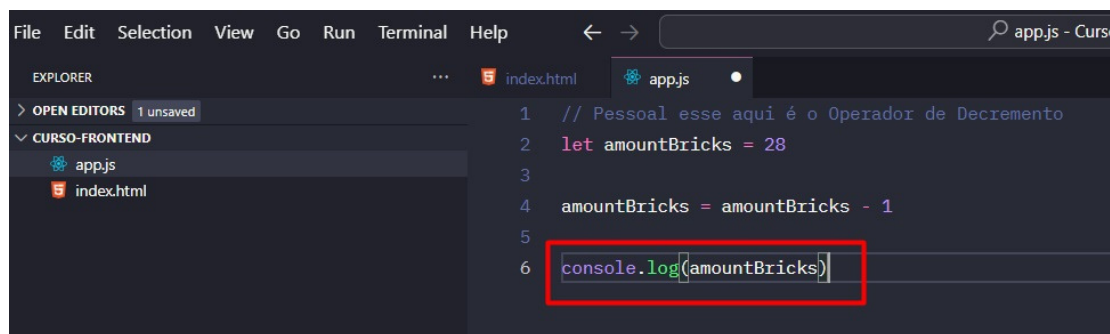
Numbers - Parte Final



```
1 // Pessoal esse aqui é o Operador de Decremento
2 let amountBricks = 28
3
4 amountBricks = amountBricks - 1
5
```

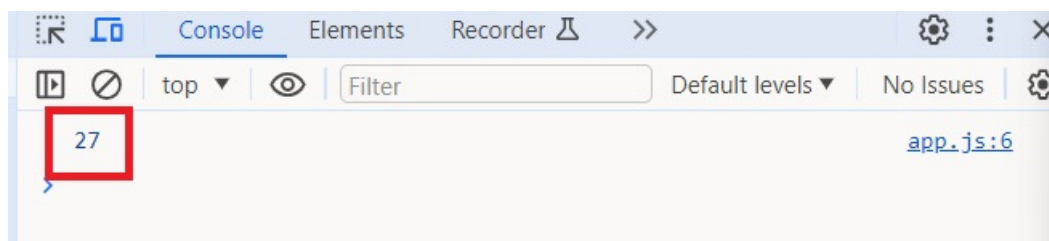
Adicionaremos um `console.log` passando a `amountBricks`. Ficando assim:

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



```
1 // Pessoal esse aqui é o Operador de Decremento
2 let amountBricks = 28
3
4 amountBricks = amountBricks - 1
5
6 console.log(amountBricks)
```

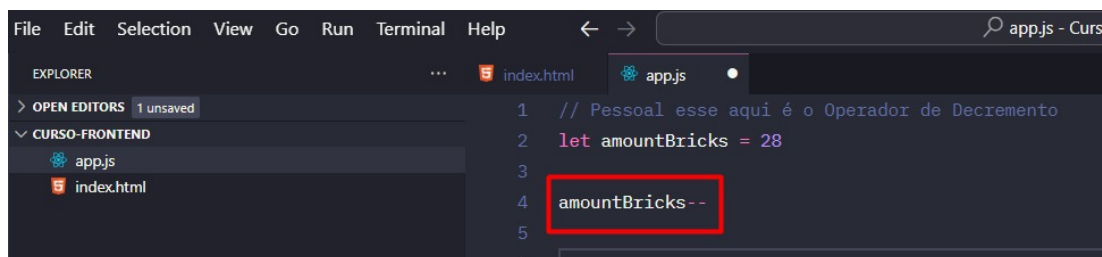
E ao salvarmos o arquivo `app.js` com aquele CTRL + S maroto podemos ver no console do Navegador(Chrome) que o número 27 é exibido.



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--`



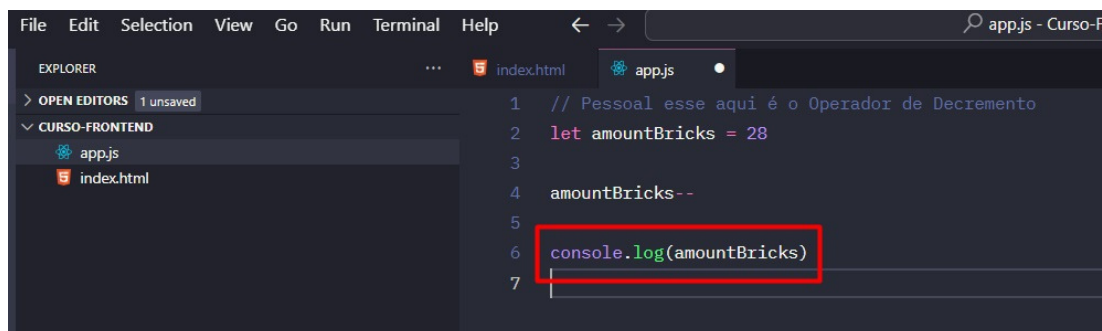
A screenshot of the Visual Studio Code editor interface. The Explorer panel on the left shows a project named 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main editor window is open to 'app.js'. The code in the editor is as follows:

```
1 // Pessoal esse aqui é o Operador de Decremento
2 let amountBricks = 28
3
4 amountBricks--
5
6
```

The line `amountBricks--` on line 4 is highlighted with a red rectangular box.

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

`console.log(amountBricks)`



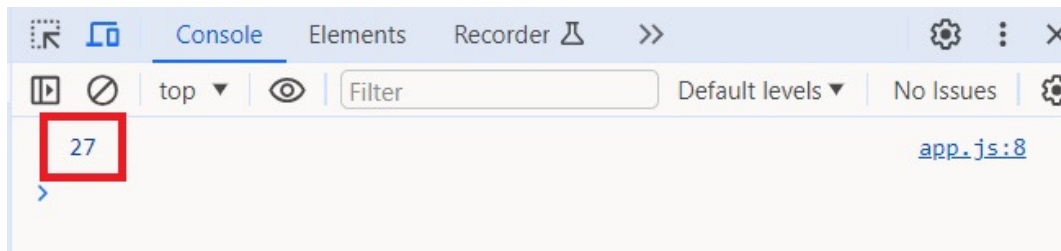
A screenshot of the Visual Studio Code editor interface, similar to the previous one. The Explorer panel shows the same project structure. The main editor window is open to 'app.js'. The code in the editor is as follows:

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

The line `console.log(amountBricks)` on line 6 is highlighted with a red rectangular box.

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

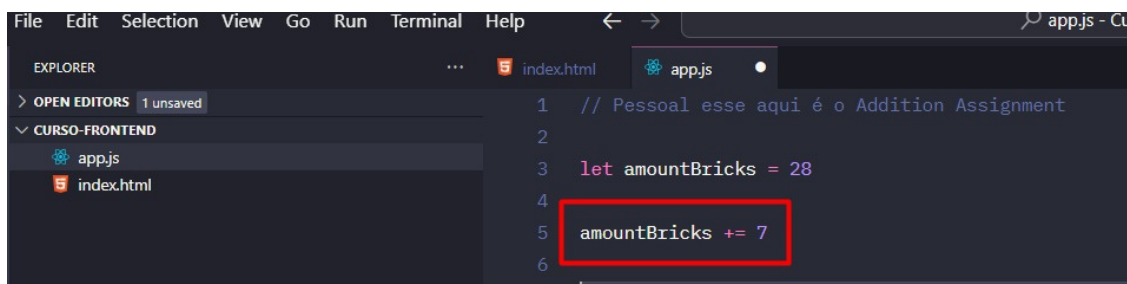
Igor, incrementamos 1 e decrementamos 1 nos exemplos anteriores, e se eu quiser adicionar mais de 1... por exemplo se eu quiser adicionar + 7 ?
Ótima pergunta, essa é a missão para o nosso amigo: O Operador Addition Assignment.

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
```

A screenshot of a code editor interface. The Explorer panel on the left shows a project named 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main editor area shows the 'app.js' file with the following code:

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

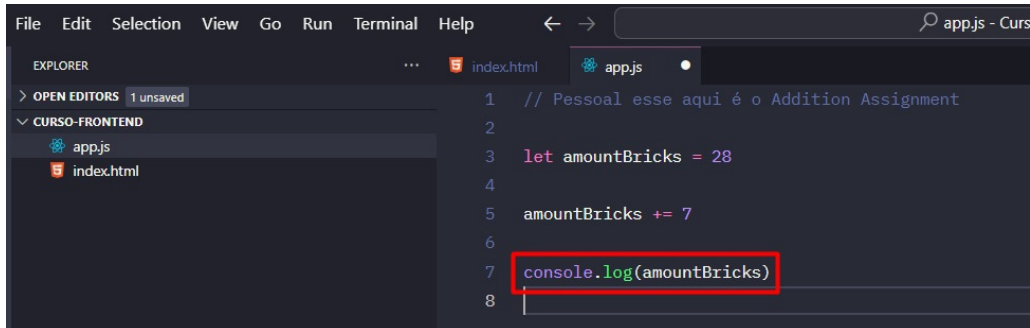
The line `amountBricks += 7` is highlighted with a red rectangular box.

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

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

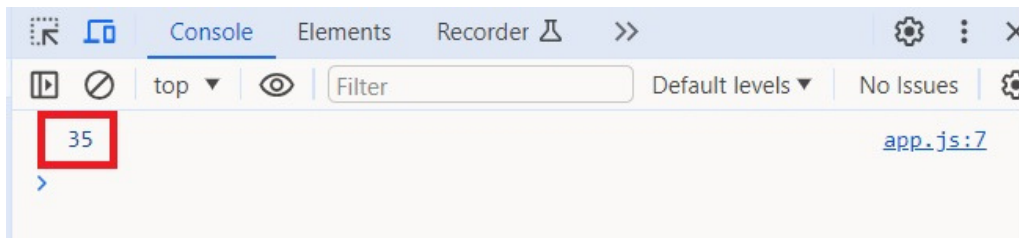


Numbers - Parte Final Addition Assignment



```
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.



Numbers - Parte Final

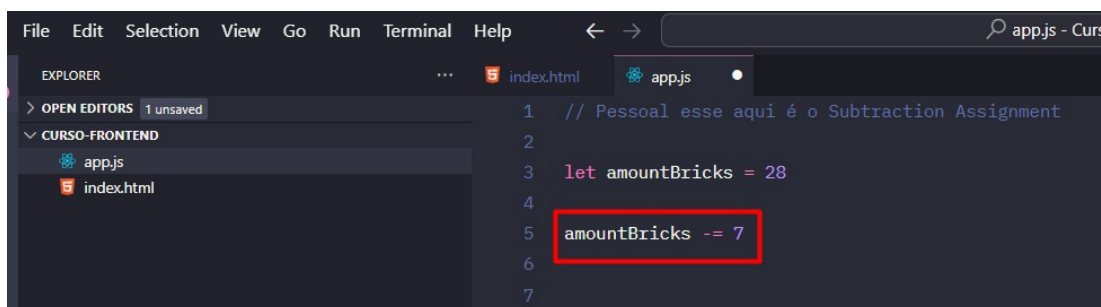
Subtraction Assignment

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
```



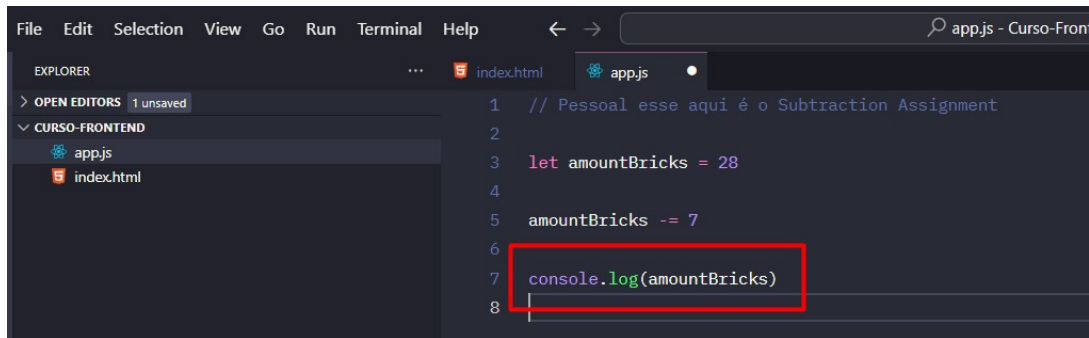
Adicionaremos um `console.log` passando a `amountBricks`. Ficando assim:

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



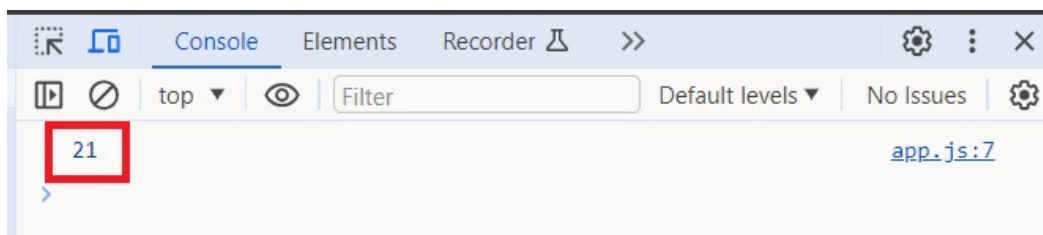
Numbers - Parte Final

Subtraction Assignment



```
1 // Pessoal esse aqui é o Subtraction 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 21 é exibido.



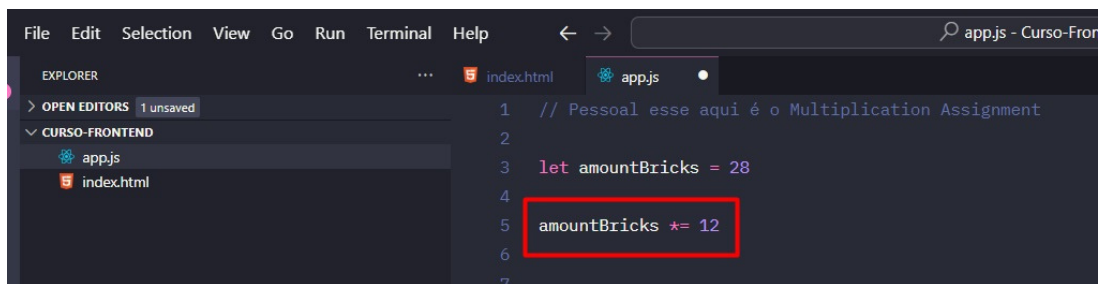
Numbers - Parte Final Multiplication Assignment

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`



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

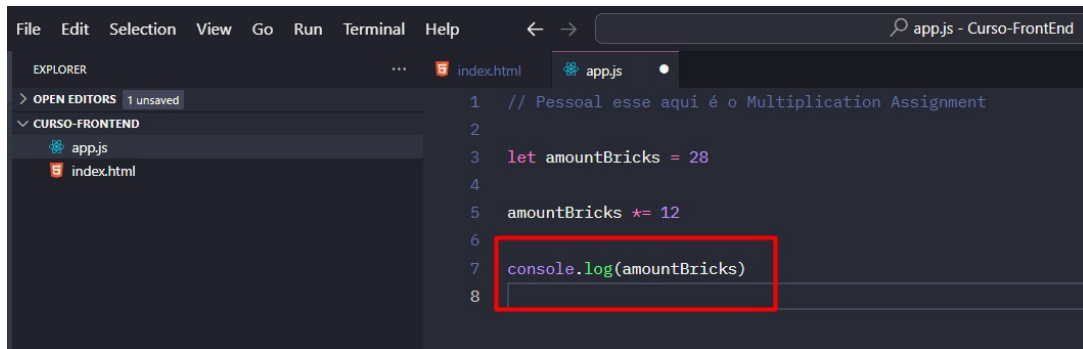
Adicionaremos um `console.log` passando a `amountBricks`. Ficando assim:

`console.log(amountBricks)`



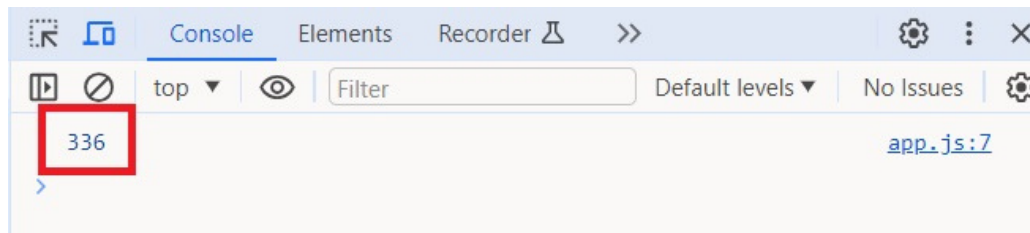
Numbers - Parte Final

Multiplication Assignment



```
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.



Numbers - Parte Final

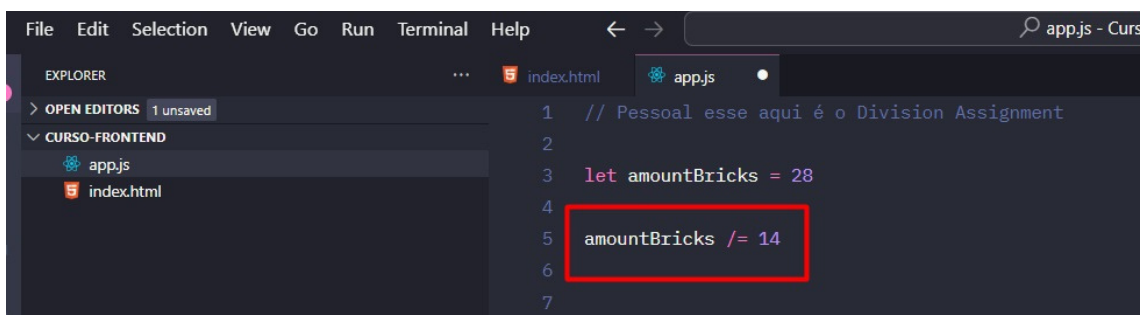
Division Assignment

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
```



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

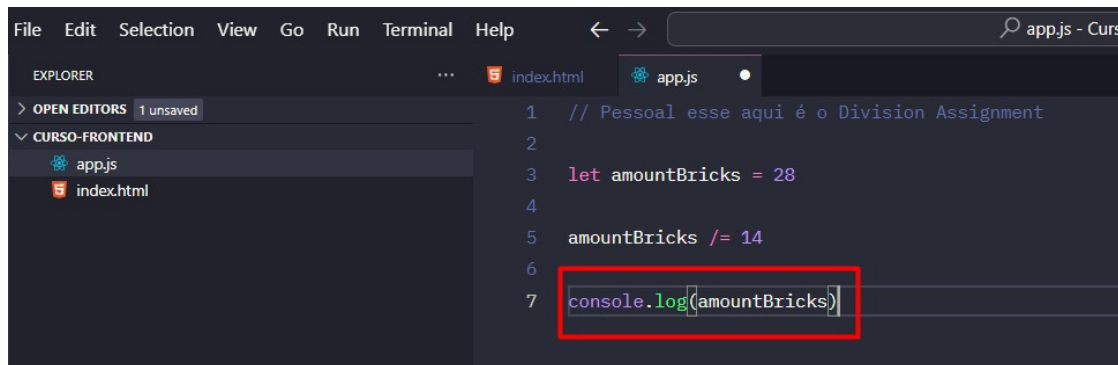
Adicionaremos um `console.log` passando a `amountBricks`. Ficando assim:

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



Numbers - Parte Final

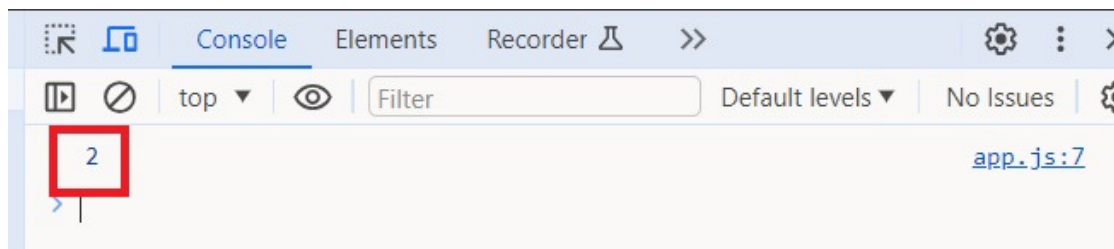
Division Assignment



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

The screenshot shows a code editor with a dark theme. The Explorer panel on the left shows a project named 'CURSO-FRONTEND' with two files: 'app.js' and 'index.html'. The main editor area shows the content of 'app.js'. The code consists of a comment, a variable declaration, an assignment, and a log statement. The log statement '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 2 é exibido.



Numbers - Parte Final

NaN - Not a Number

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

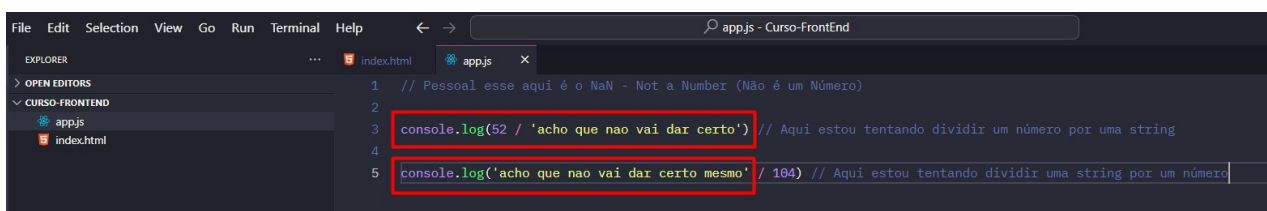
Dentro do nosso arquivo app.js, bora declarar um comentário: `// Pessoal esse aqui é o NaN - Not a Number (Não é um Número)`

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 `console.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)`

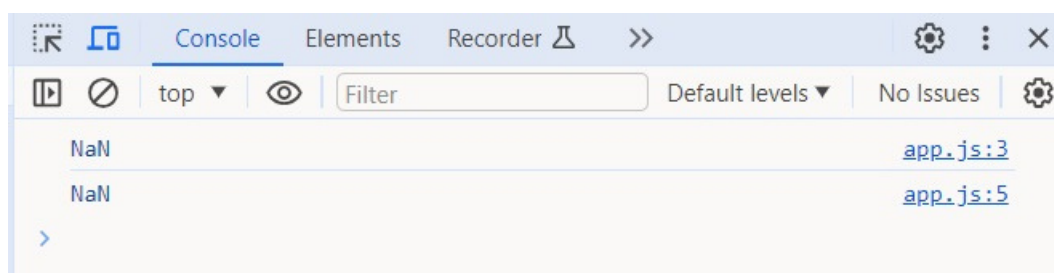


```
1 // Pessoal esse aqui é o NaN - Not a Number (Não é um Número)
2
3 console.log(52 / 'acho que nao vai dar certo') // Aqui estou tentando dividir um número por uma string
4
5 console.log('acho que nao vai dar certo mesmo' / 104) // Aqui estou tentando dividir uma string por um número
```

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 início, 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:

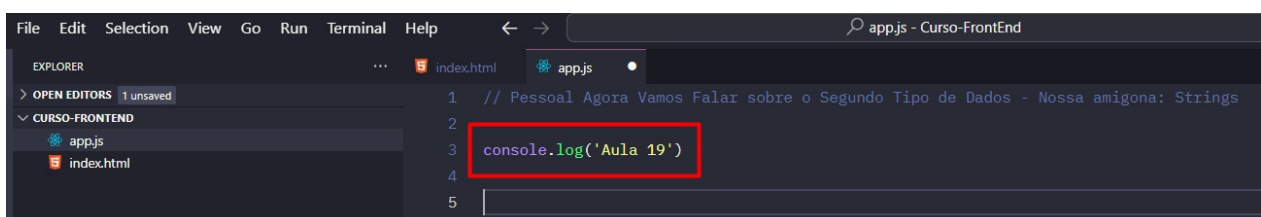
Igor, uma dúvida.... Por que nós usamos Strings?

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 lembramos 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')
```

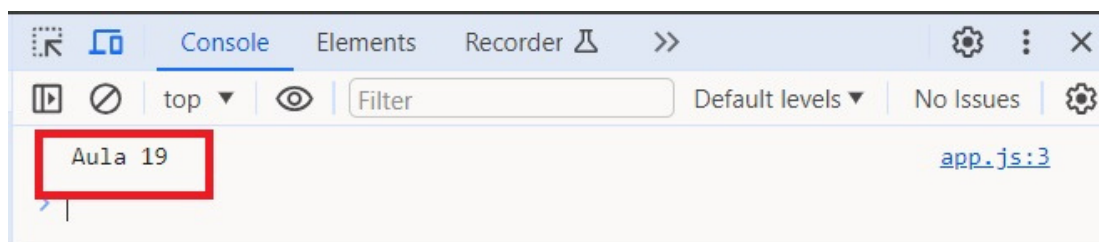


```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2
3 console.log('Aula 19')
4
5
```



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?

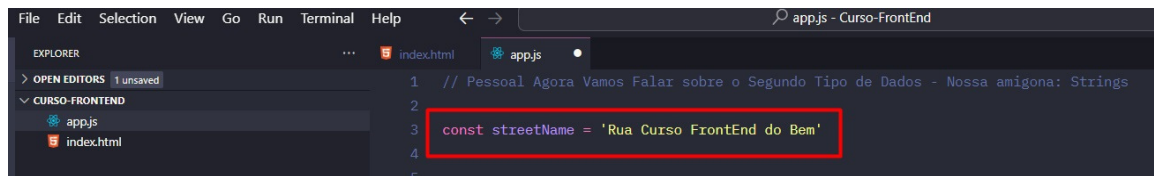
As nobres aspas aparecem apenas dentro do VSCode para que o mesmo consiga identificar que ali é uma string, e a partir do momento que é renderizado/mostrado no navegador as aspas não são mais necessárias mostrando apenas o conteúdo dentro dela (isso vale tanto para aspas duplas quanto para aspas simples).

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



```
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
```

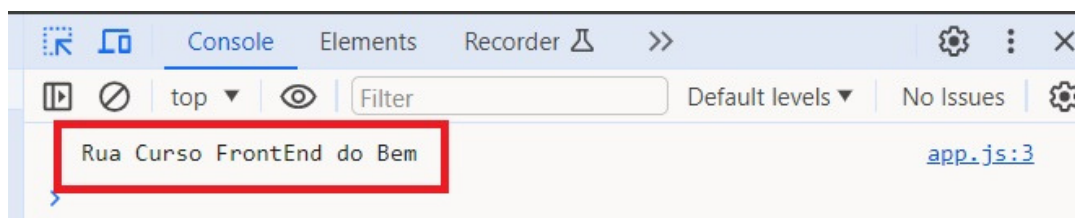
Podemos exibir essa constante no console.log passando a const streetName entre parênteses. Ficando assim:

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



```
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
```

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.



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'
```

A screenshot of a code editor interface. The Explorer panel on the left shows a project named 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main editor area shows the content of 'app.js' with the following code:

```
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
```

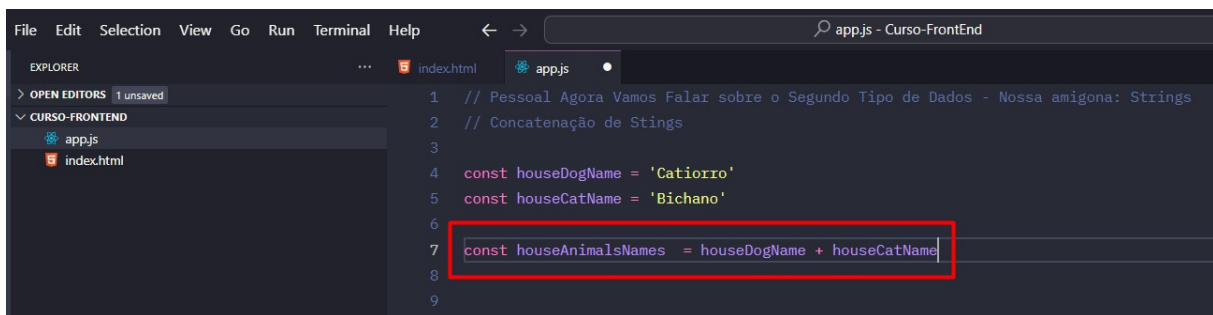
The lines 4 and 5 are highlighted with a red rectangular box.

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
```



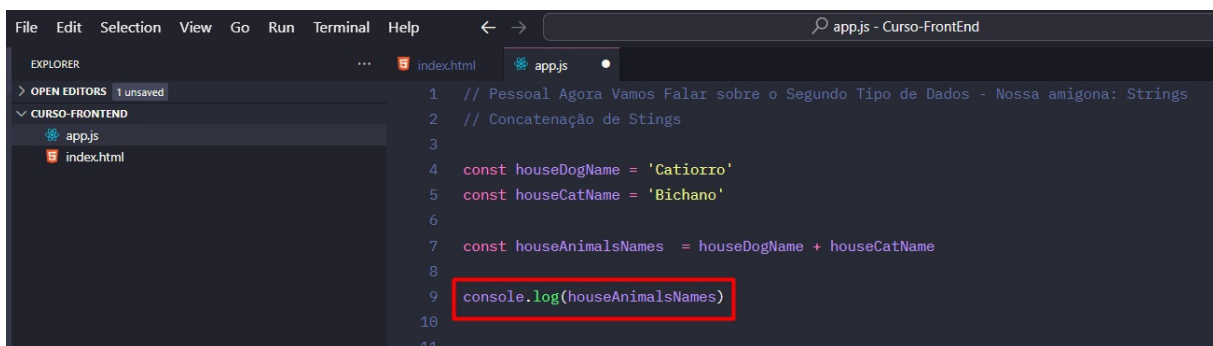
The screenshot shows a VS Code editor window with the file `app.js` open. The code contains the following lines:

```
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 box.

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

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



The screenshot shows the same VS Code editor window with the file `app.js` open. The code now includes an additional line:

```
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
```

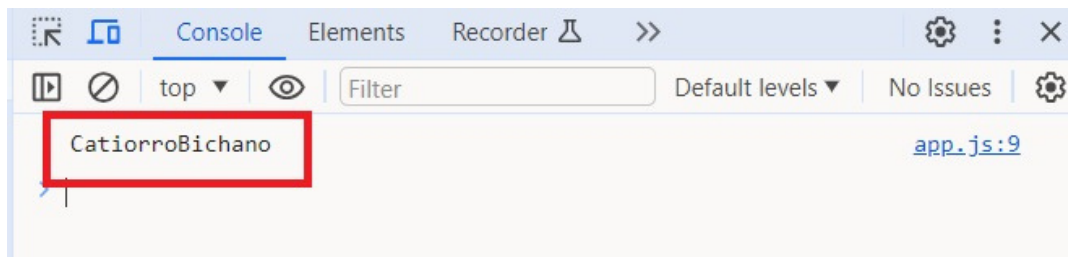
The line `console.log(houseAnimalsNames)` is highlighted with a red box.



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.

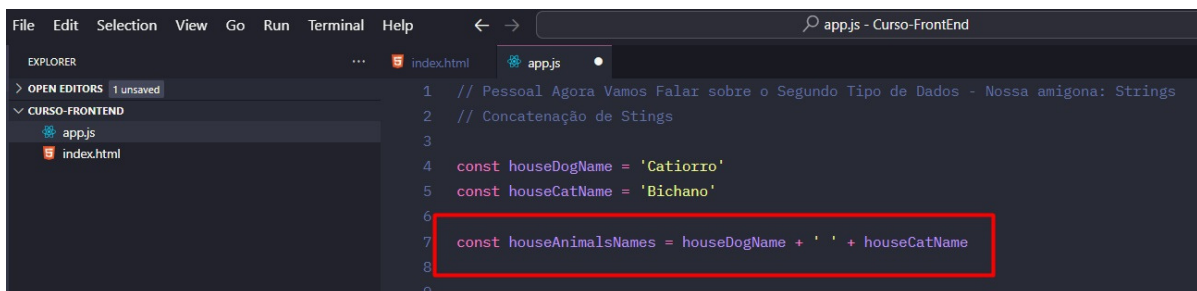


Notem que aqui não existe um espaço entre as strings (ficou tudo grudado... "sai pra lá bichano", "sai você pra lá catiorro"... calma meus amiguinhos não briguem já resolvemos isso).

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 contem um espaço em branco e o houseCatName). Ficando assim:



Strings

Concatenação de Strings

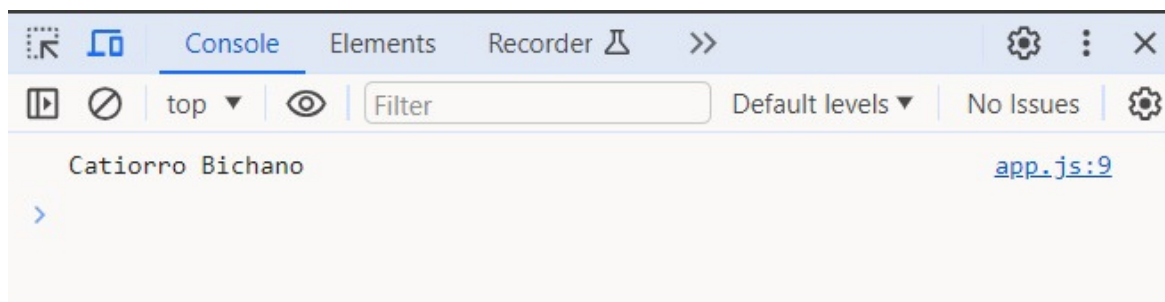
Adicionaremos um `console.log` passando a `houseAnimalsNames` . Ficando assim:

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



```
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
```

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.



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])
```



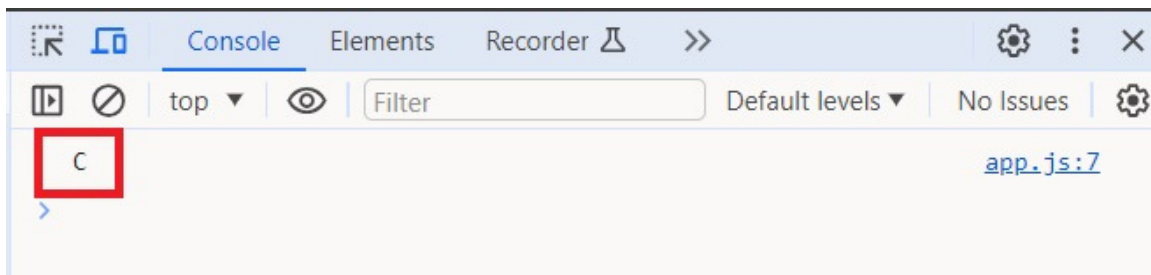
Strings

Como acessar Caracteres



```
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[0])
8
```

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.



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])
```

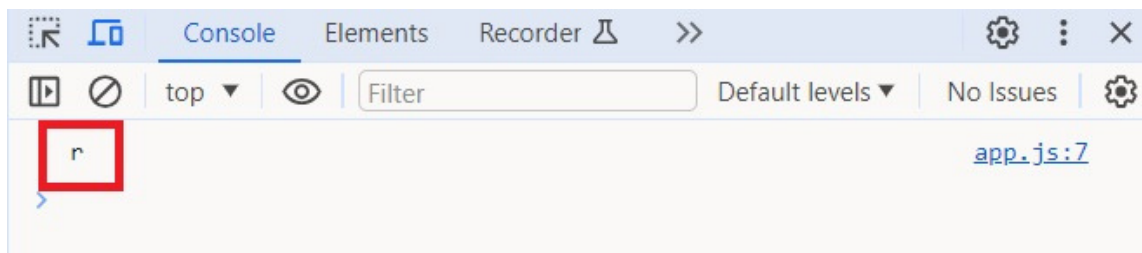
Strings

Como acessar Caracteres



```
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.



Strings

Comprimento de uma String

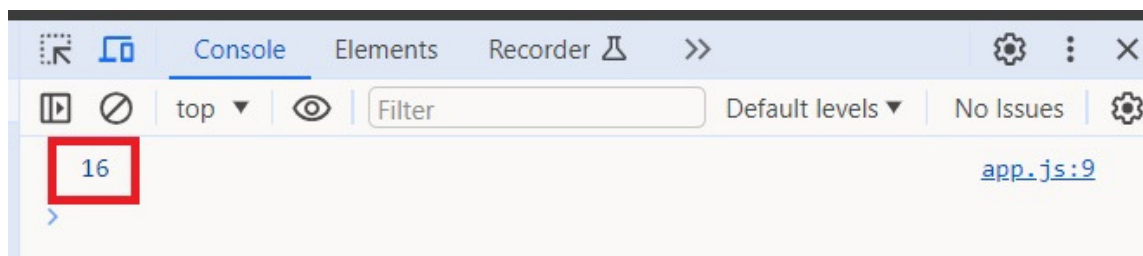
As nossas amigonas Strings também possuem propriedades e métodos. É uma das propriedades que uma string conta é o `length`, o `length` é uma propriedade que vai nos informar o comprimento de uma string ou seja quantos caracteres ela tem. Declaremos um `console.log` que vai exibir `houseAnimalsNames` 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)
```



```
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.length)
10
```

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.



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!!!



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())
```

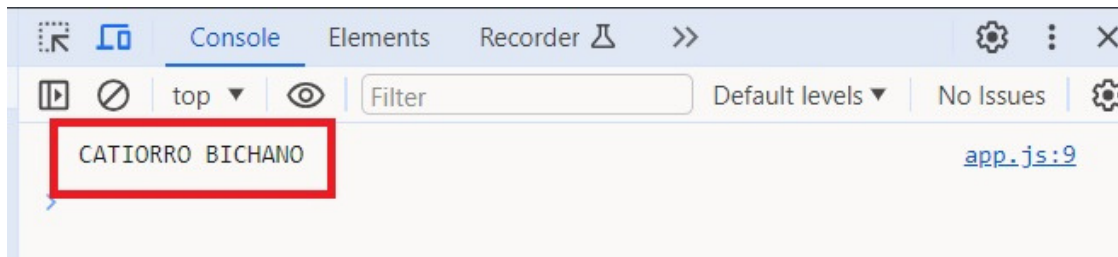


Se a gente salvar esse arquivo, a gente vai ver no console que esse metodo pegou a string Catiorro Bichano e fez com que todos os caracteres dessa string ficassem em maiusculos.



Strings

Métodos de uma String

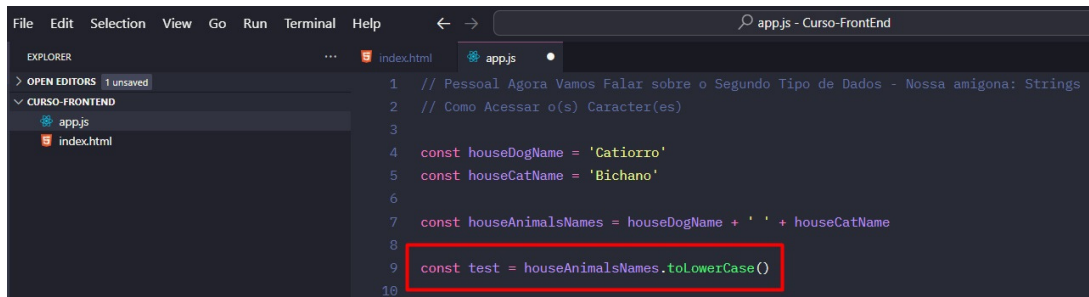


Tem um outro método que gostaria de mostrar para o conhecimento de vocês. Mas/porem/entretanto/todavia antes abaixo desse `console.log(houseAnimalsNames.toUpperCase())`, declaremos uma `const test` que receberá o resultado de `houseAnimalsNames.toLowerCase()` e como você deve ter pensado o método `toLowerCase()` 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



```
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
```

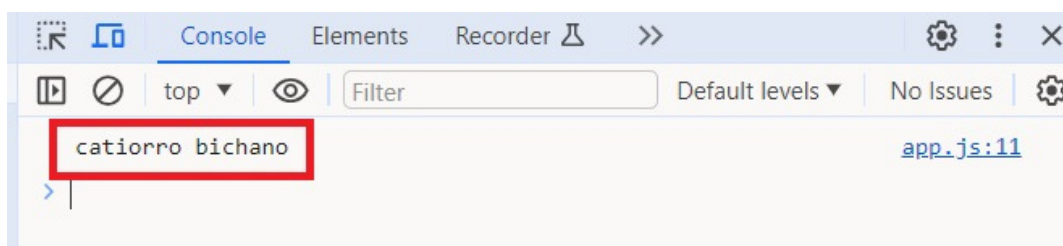
Vamos declarar um console.log que exhibe a test. Ficando assim:

`console.log(test)`



```
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)
12
```

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.



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

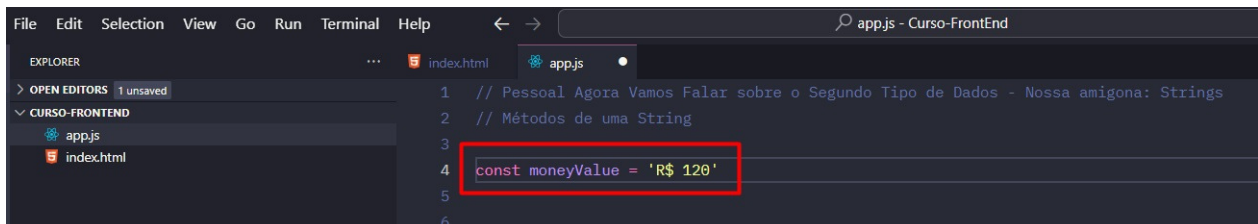
Métodos de uma String

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'
```



The screenshot shows a code editor with a dark theme. The Explorer panel on the left shows a project named 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main editor area shows the content of 'app.js' with the following code:

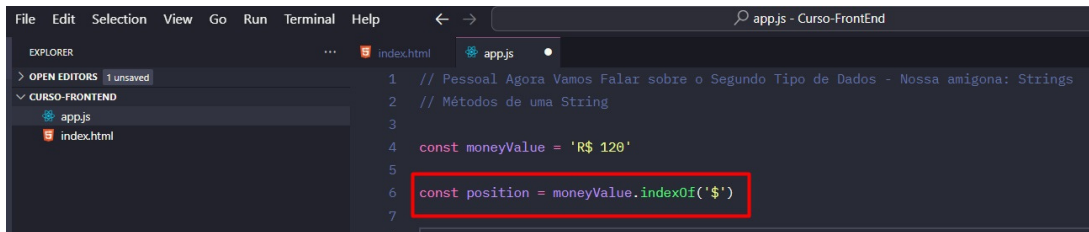
```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Métodos de uma String
3
4 const moneyValue = 'R$ 120'
5
6
```

The line `const moneyValue = 'R$ 120'` is highlighted with a red rectangular box.

Strings

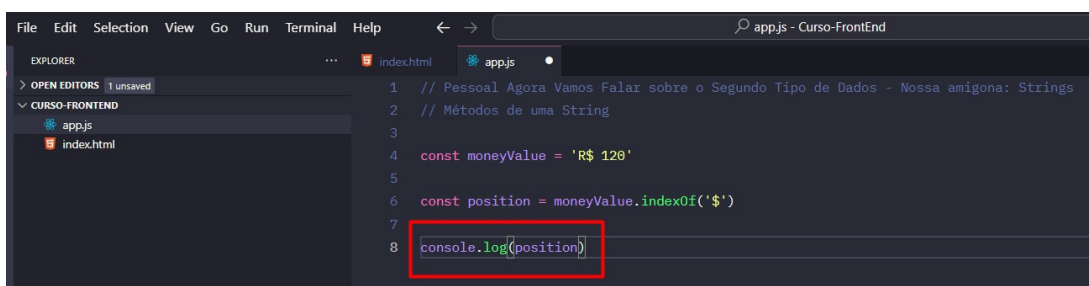
Métodos de uma String

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



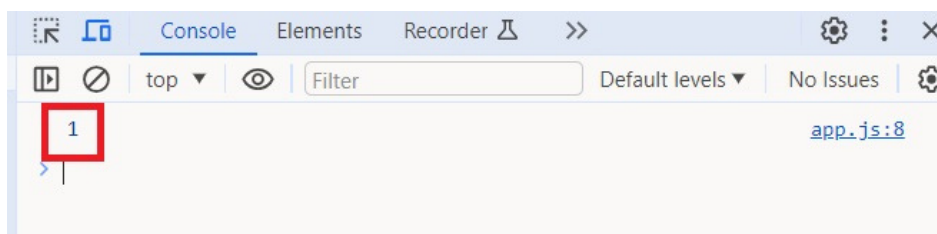
```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amigona: Strings
2 // Métodos de uma String
3
4 const moneyValue = 'R$ 120'
5
6 const position = moneyValue.indexOf('$')
7
```

Vamos declarar um `console.log(position)`



```
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 vê no console que o número 1 será exibido.



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.



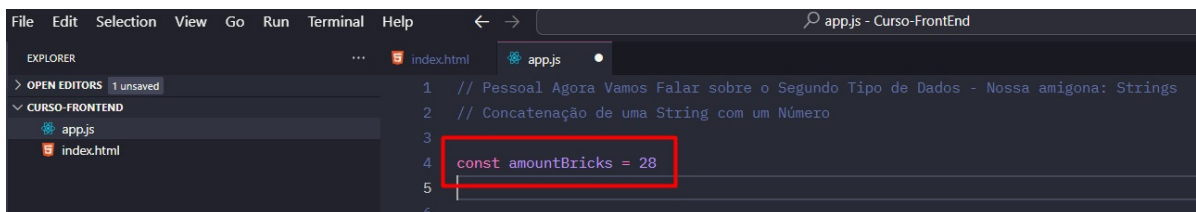
Strings

Concatenação de String com Número

Podemos concatenar string com o operador +, é possível fazer algo parecido também com o número se tentarmos adicioná-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
```

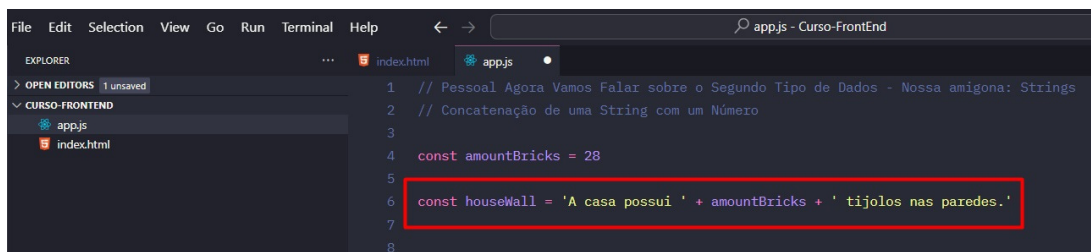


```
1 // Pessoal Agora Vamos Falar sobre o Segundo Tipo de Dados - Nossa amiga: Strings
2 // Concatenação de uma String com um Número
3
4 const amountBricks = 28
5
6
```


Strings

Concatenação de String com Número

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



A screenshot of the Visual Studio Code editor interface. The Explorer panel on the left shows a project named 'CURSO-FRONTEND' with files 'app.js' and 'index.html'. The main editor window displays the 'app.js' file. The code in the editor is as follows:

```
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.'
```

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

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



A screenshot of the Visual Studio Code editor interface, similar to the previous one. The main editor window displays the 'app.js' file. The code in the editor is as follows:

```
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.'
```

The line `console.log(houseWall)` on line 8 is highlighted with a red rectangular box.

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

Strings

Concatenação de String com Número



Sugestões de prática para essa aula:

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!!!!



O que vamos aprender na aula 20:

Na vigésima aula do curso(Sexta aula de JavaScript) o que veremos: É surpresa!!!

Feedbacks dessa décima nona aula são bem-vindos.

Nos vemos na décima nona aula que será disponibilizada no dia 04/04/2024 às 8h30 (Horário de Brasília) – Dentro do meu perfil no LinkedIn.



/igor-rebolla