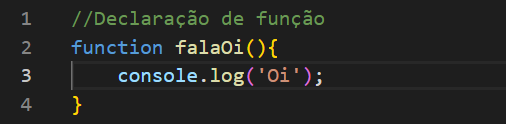
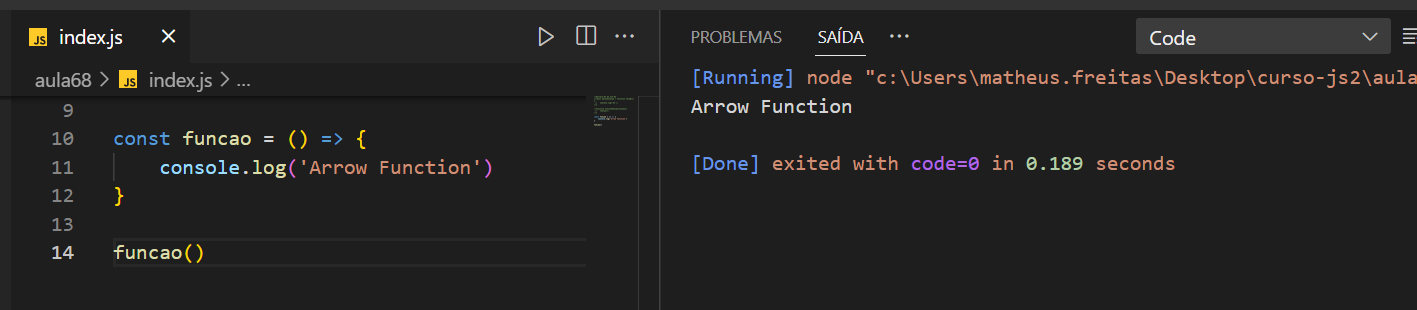
**AULA 68**

* Aula sobre funções
* Modos de declarar uma função
  + Declaração normal

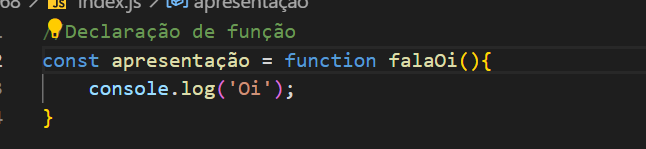


Quando declaramos a variável desta forma, acontece o evento Function hoisting, que dá prioridade a execuções declaradas desta for ou variáveis declaradas com var

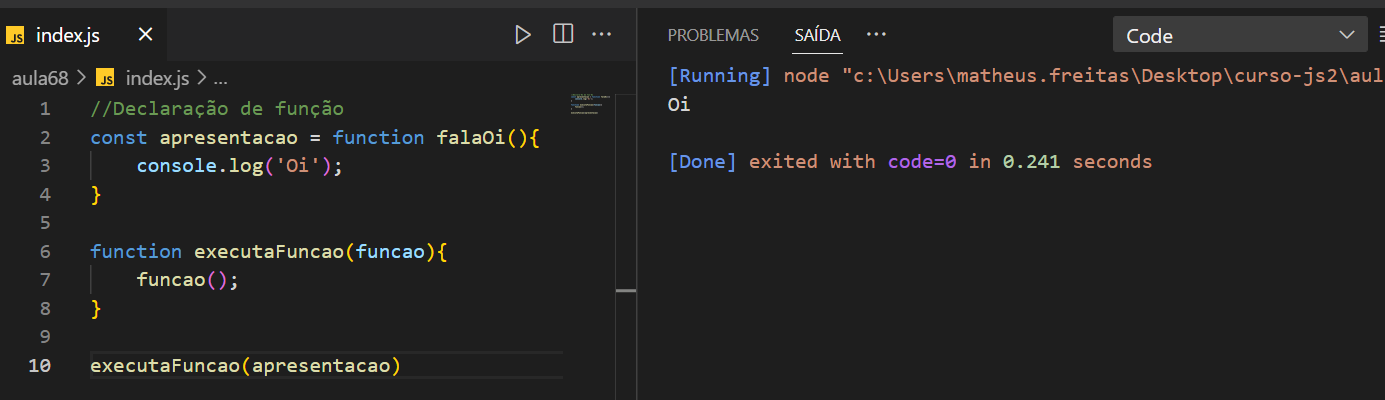
* + Arrow Function



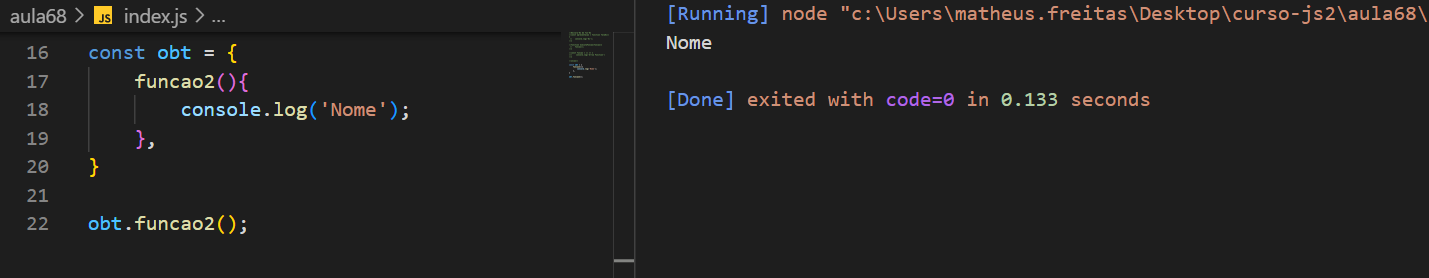
* Funções são objetos de primeira classe, ou seja, podem ser utilizados como dados. Também conhecida como Function Expression. Exemplo:



* Como podem ser declaradas como variáveis, também podemos utiliza-las como parâmetros para outras funções.

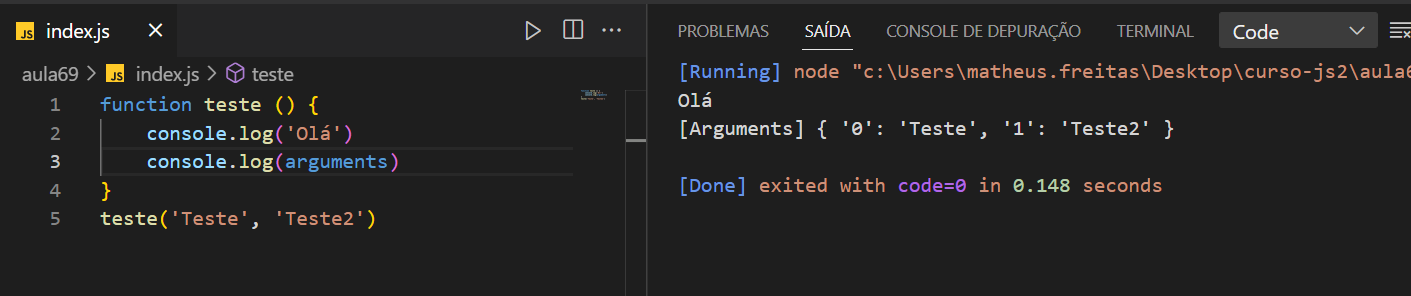


* Também podemos declarar uma função dentro de um objeto, criando um método nos atributos

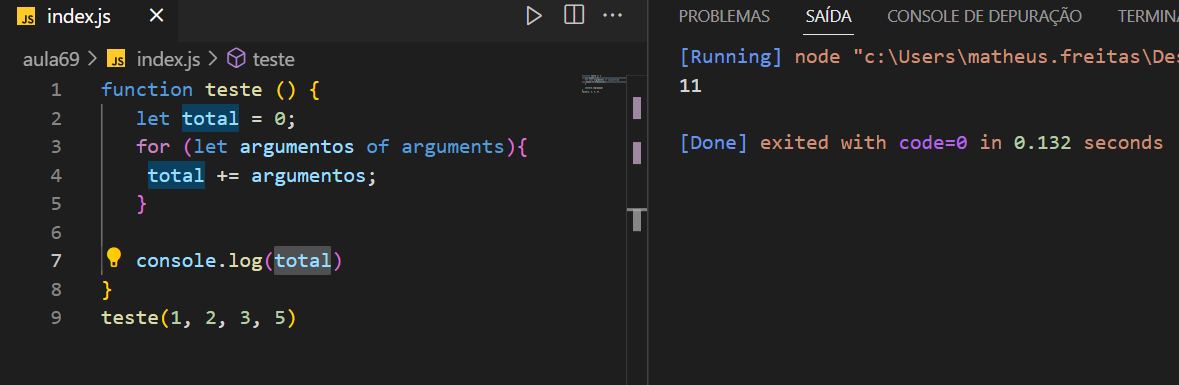


**AULA 69**

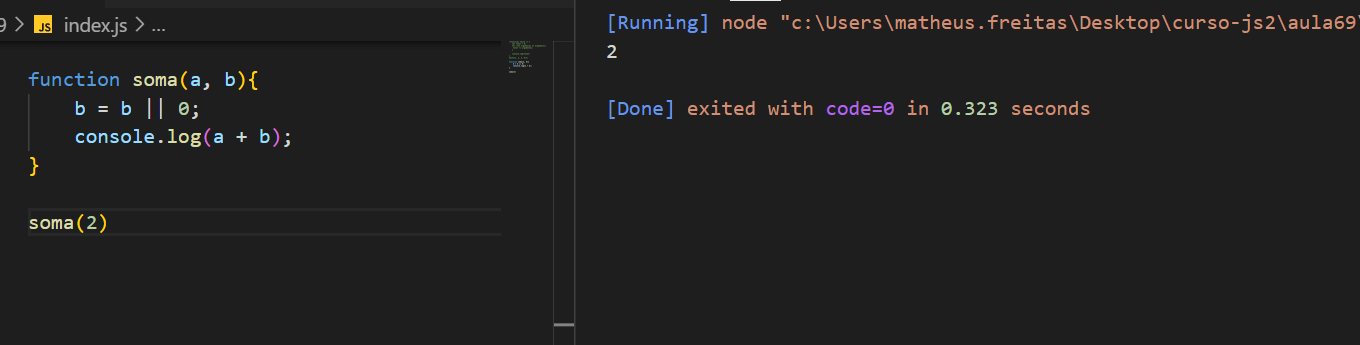
* Aula sobre parâmetros
* Uma função pode ou não ter parâmetros
* Nos casos onde você não declara que precisa de um parâmetro, porém ao ser acionada a função você declarar um parâmetros, ele irá armazenar os dados dentro de um objeto chamado arguments, então caso você queria exibir esses valor, é necessário utilizar essa variável chave



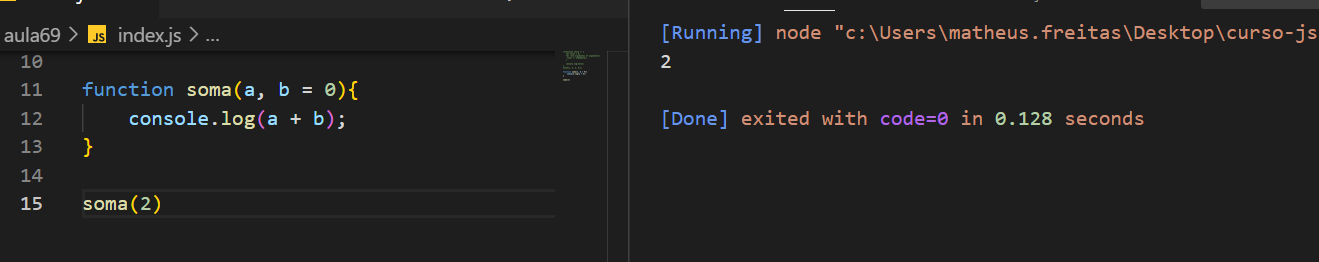
* Podemos utilizar esses valores armazenados dento do arguments, fazendo cálculos ou manipulando eles



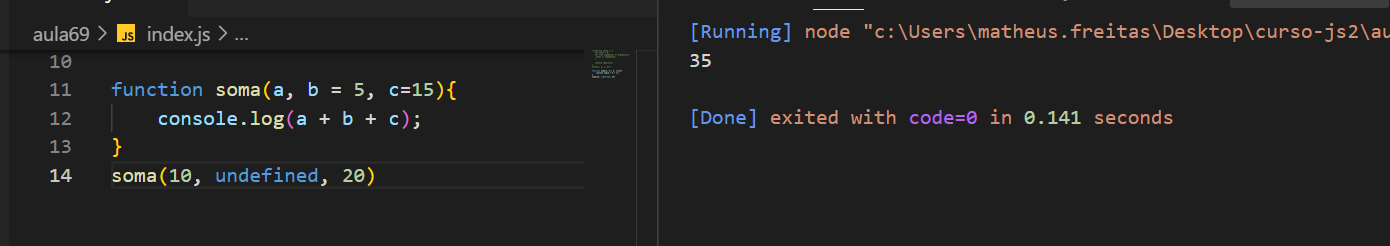
* Podemos definir um valor padrão de duas formas. Exemplo:
  + Primeira:



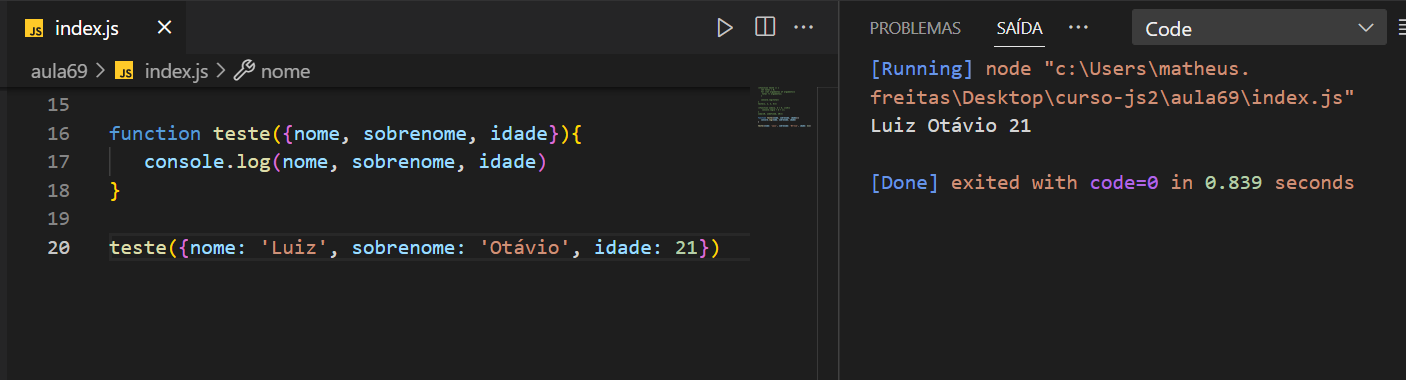
* + Segunda



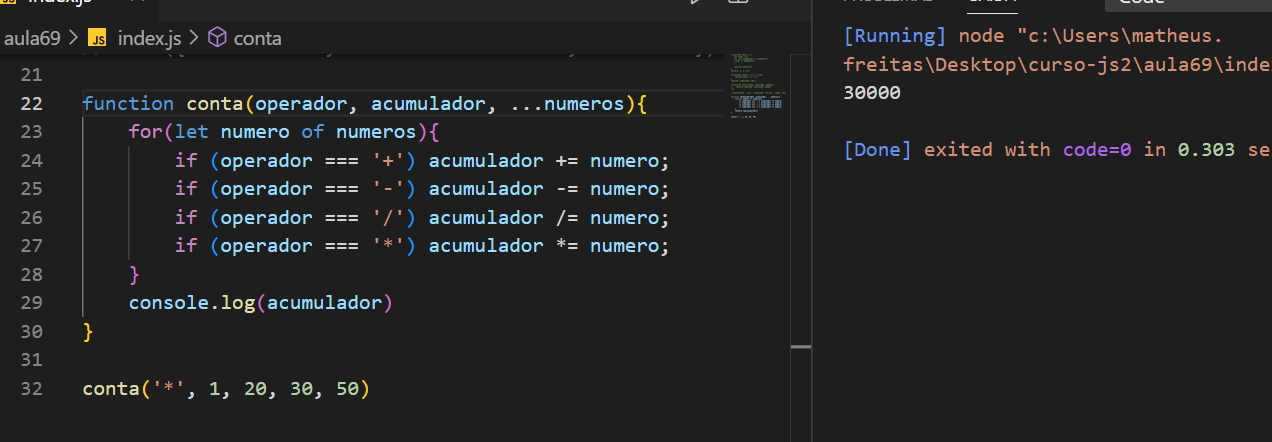
* Caso você queria utilizar um valor padrão de variável que fica no meio de outros dois parâmetros, precisamos utilizar o valor undefined. Exemplo:



* Podemos atribuir os parâmetros por atribuição por desestruturação



* Podemos utilizar o operador de resto (...) para atribuir ao ultimo parâmetro. Exemplo:



**AULA 70**

* Aula sobre o comando return
* O comando retorna um valor gerado pela função
* O comando encerra a execução da função

**AULA 71**

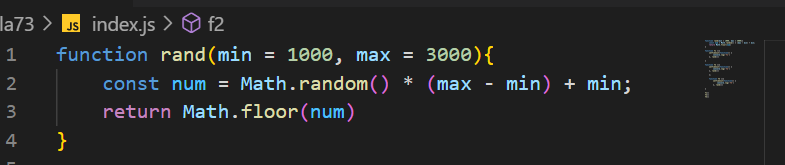
* Aula sobre escopo léxico
* Variáveis globais e locais

**AULA 72**

* Aula sobre clousers
* É muito relacionado ao escopo léxico das funções
* Clousers são os elementos que uma função utiliza que está fora do seu escopo
* Podem ser uma variável ou até mesmo um parâmetro

**AULA 73**

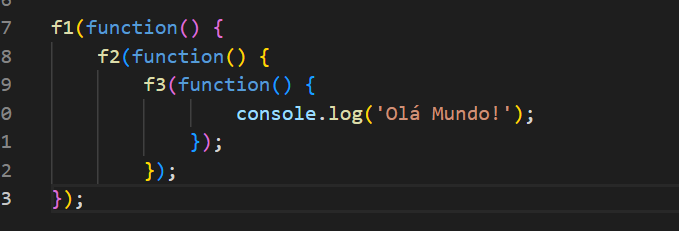
* Aula sobre funções de call-back
* São funções acionadas quando ocorrem alguma ação é finalizada
* Criamos um execicio onde criamos 3 funções, que cada um recebia um tempo para ser executado, que vinha de outra função que cria números aleatoriamente entre 1000 e 3000
  + Função para criar um timer:



* Funções que recebem os números aleatórios e executam de acordo com o tempo recebido



* Para garantir que as funções sejam executadas pela sua ordem numérica, utilizamos funções de call-back
* Acionamos a função f1, criando dentro dela uma função que aciona a f2, e dentro da função que executa a f2 uma que acione a f3 (também conhecido como callback hell)

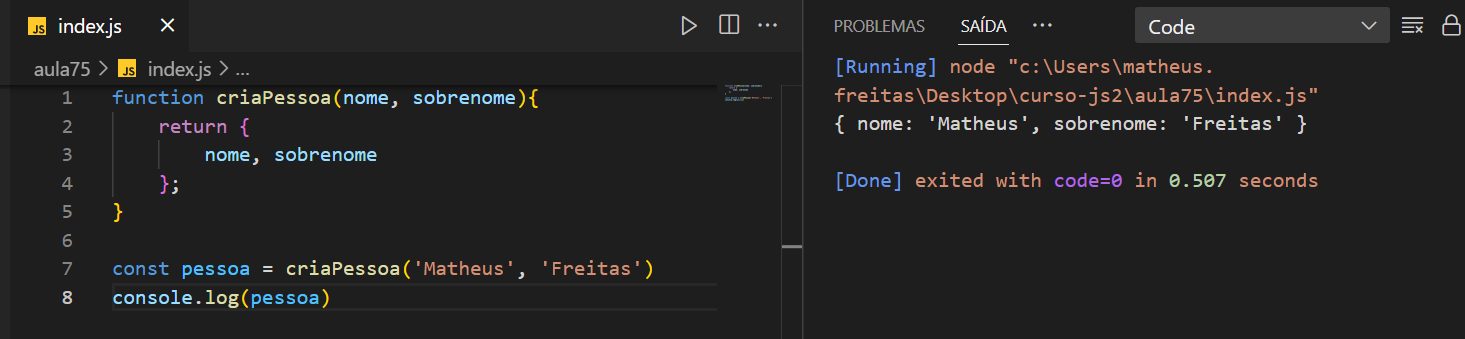


**AULA 74**

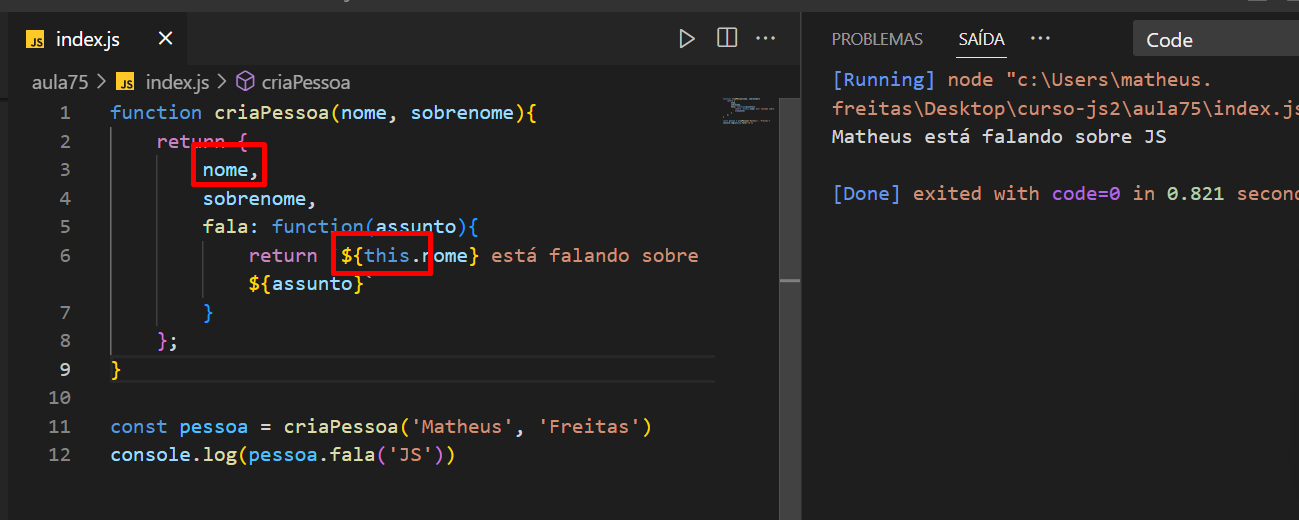
* Aula sobre funções imediatas (IIFE – Immediately Invoked Function Expression)
* É uma função que é acionada automaticamente
* Ela não realizar alterações em escopos globais
* Criamos um exercício utilizando uma função imediata, criando dentro dela subfunções
* Como as outras funções, as variáveis criadas nelas não podem ser alteradas pelo escopo global

**AULA 75**

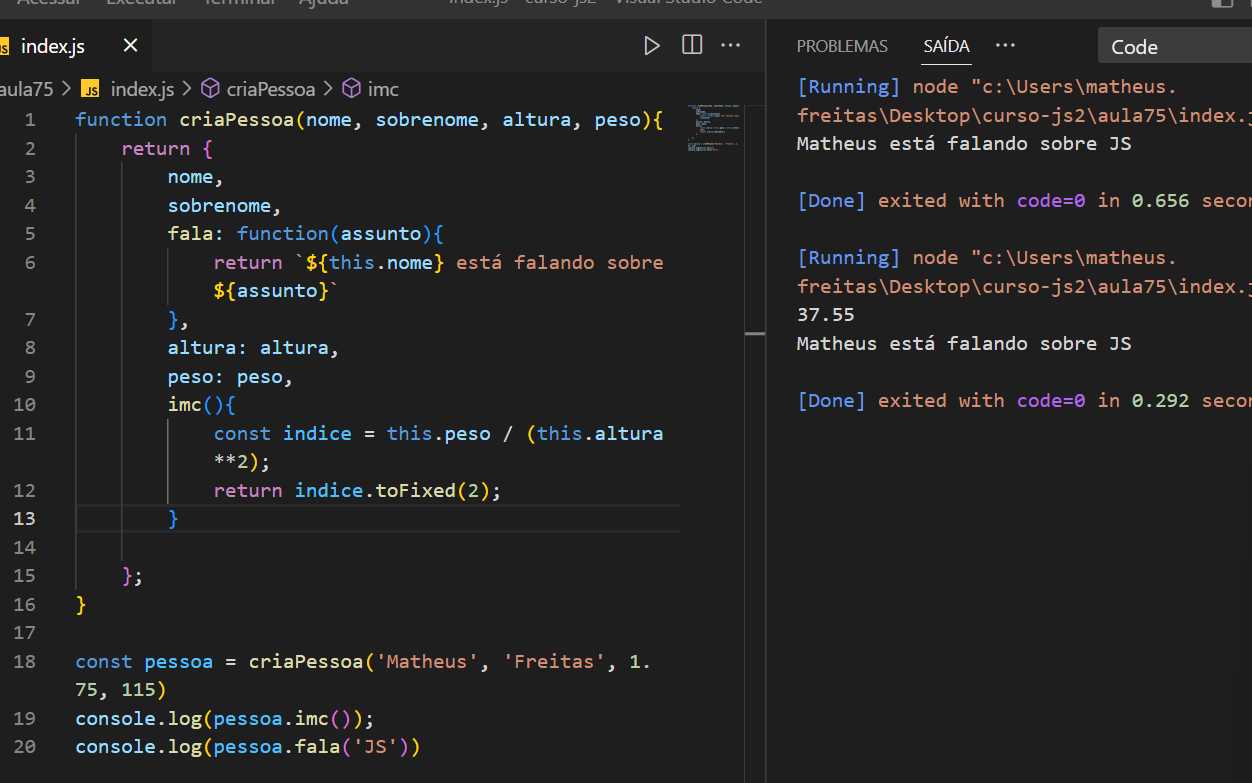
* Aula sobre funções fabrica
* São funções que criam objetos. Exemplo:

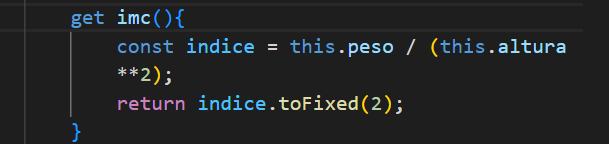


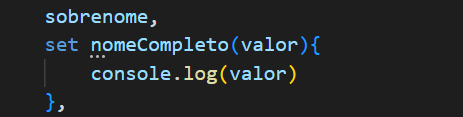
* O comando This serve para você acessar um parâmetro que está dentro de uma função ou elemento “pai”. Exemplo:



Criamos um exercício onde uma função cria um objeto que possui dois métodos, um para criar uma linha de texto e outra para calcular o IMC de uma pessoa

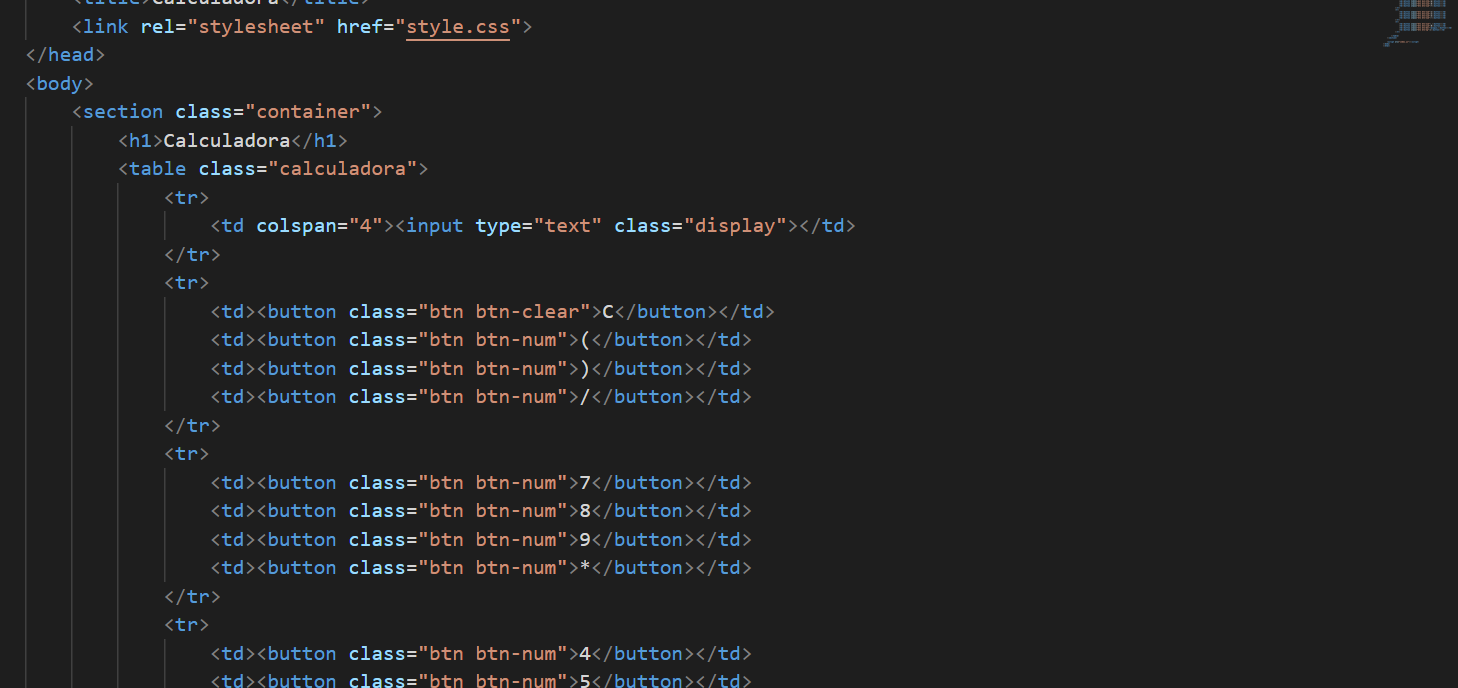


* Podemos fazer um método ser executado como um atributo, utilizando comando get
* No caso de precisar inferir os dados dentro de um método, utilizamos o comando set

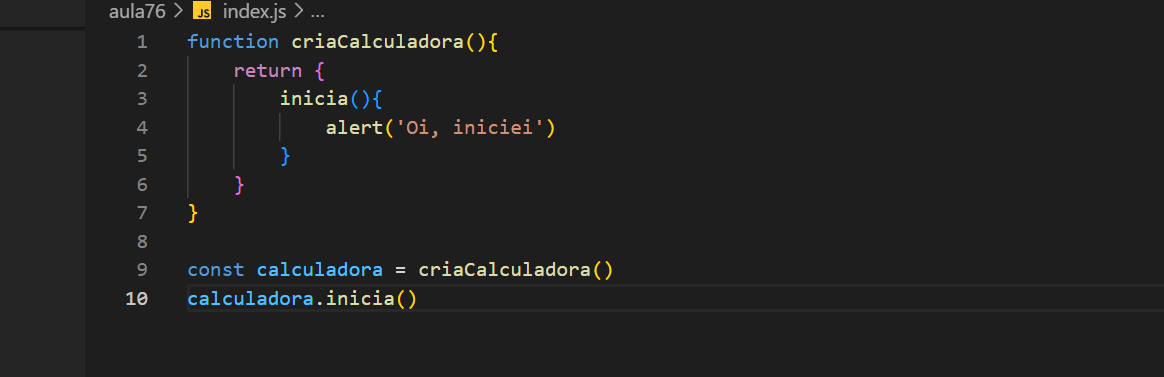


**AULA 76**

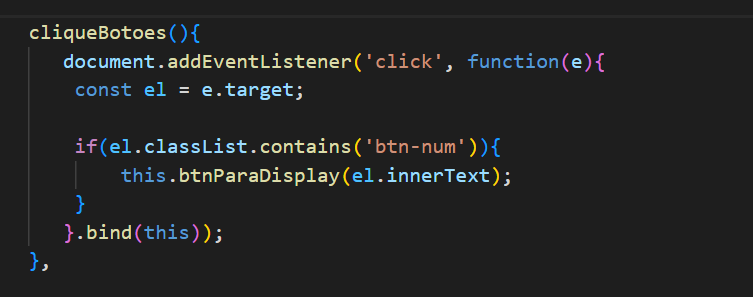
* Exercício sobre funções factory
* Criamos uma página HTML e um arquivo CSS, utilizando um titulo e uma tabela para simular a aparência de uma calculadora



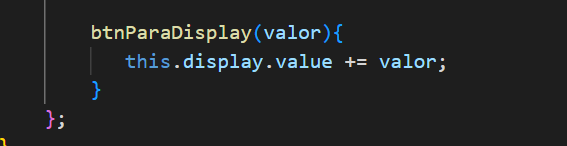
* No JavaScript, criamos uma função fabrica (função que cria um objeto) e dentro dela, criamos um método que inicia a calculadora



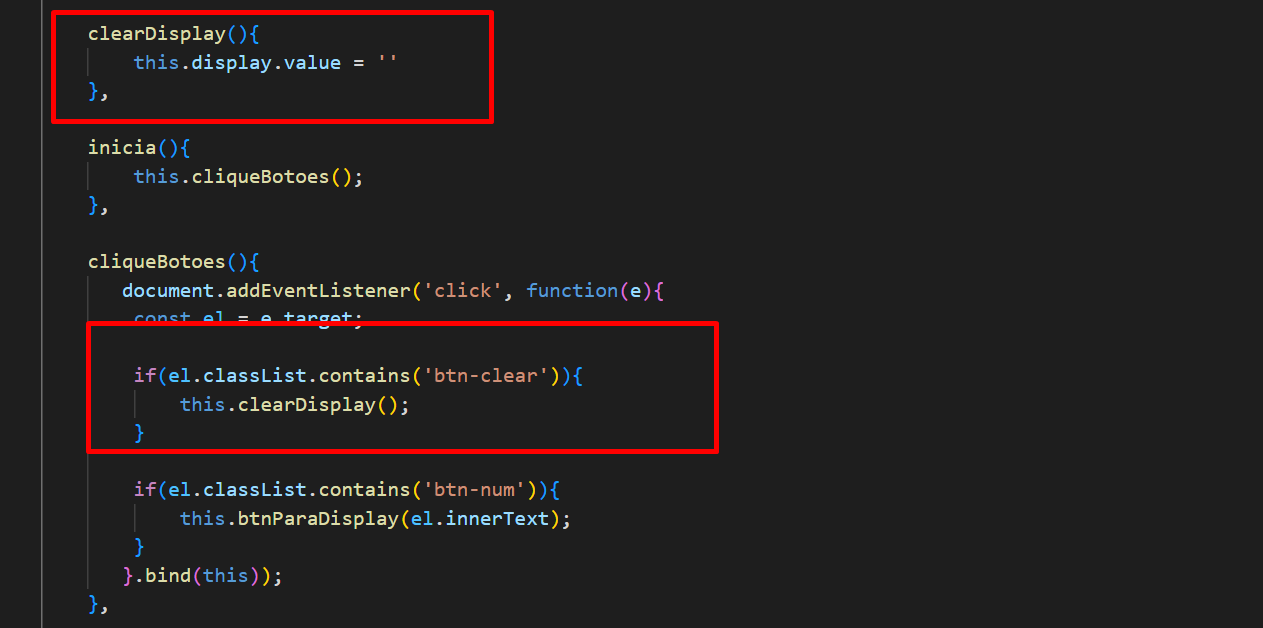
* Criamos outros dois métodos dentro do nosso objeto. O primeiro captura a ação de clicar, e se o objeto clicado tiver a classe “btn-num” o valor desse objeto é atribuído ao campo display



* O outro método faz a ação de concatenar o seu valor ao valor recebido da função anterior

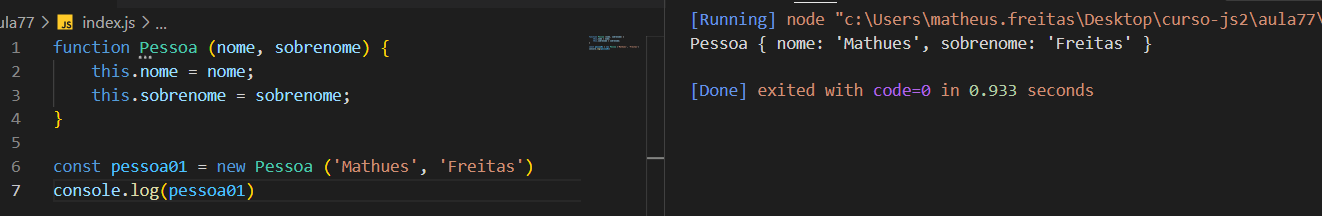


* Criamos um método que realizar a limpeza do campo display, após clicarmos o botão ‘’C’’. Atribuímos um valor de string vazio caso o botão clicado tenha a classe ‘’btn-clear’’



**Aula 77**

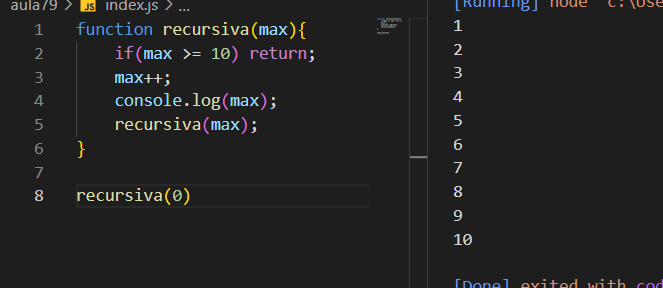
* Funções construtoras
* Assim com funções fabricas, também constroem objetos (funções fabricas são funções normais que nos retornam um objeto, já uma função construtora tem muitas funções automatizadas)
* Toda função construtora começa a letra maiúscula
* Possuem sempre a palavra “new” atribuída a ela
* O corpo da função construtora será o objeto que irá ser retornado
* Uma função construtora é apenas um molde, qual utilizamos para criar um objeto. Exemplo:



* Assim como em outras funções, podemos criar atributos privados (variáveis disponíveis apenas dentro da função)

**AULA 79**

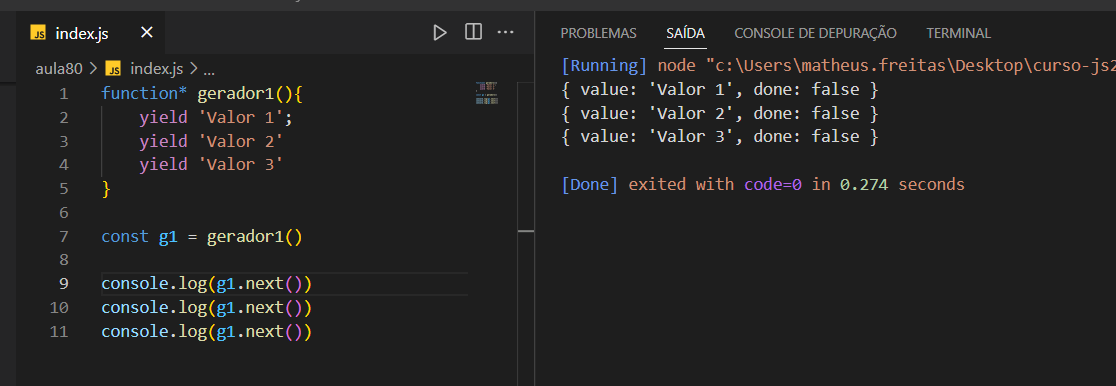
* Funções recursivas
* São funções que se auto acionam
* O motor do JavaScript pode travar uma função recursiva se ela for acionada muitas vezes
* Exemplo:



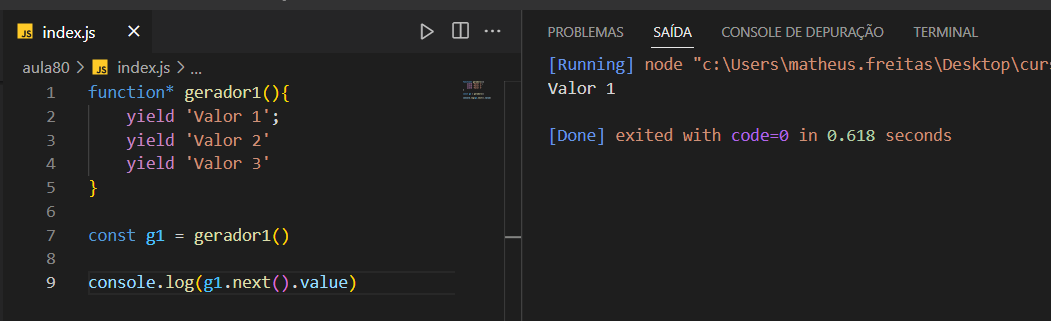
* Sempre deve ter uma valvula de escape (limite de acionamento da função)

**AULA 80**

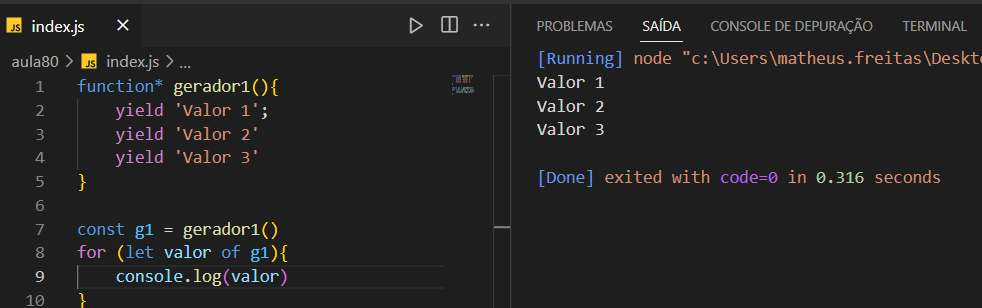
* Funções geradoras
* Essas funções possuem uma habilidade especial de "pausar o código"
* Essa função executa os seus retornos em ordem de execução, onde cada linha onde é acionada retorna apenas uma das linhas de retorno. Exemplo:



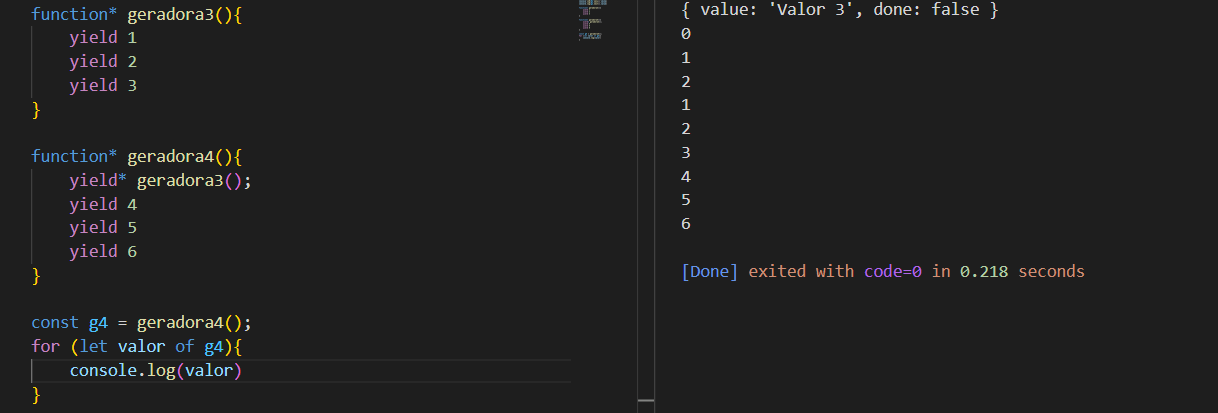
* No painel do Node, podemos ver que ele nos entrega um objeto, onde possui dos atributos
  1. Valor: Ele vai entregar o valor do yield da sequencia
  2. Done: Nesse atributo ele vai apontar false caso o yield não seja o ultimo da sequencia
* Para vermos apenas o valor do yield, utilizamos o comando .value. Exemplo:



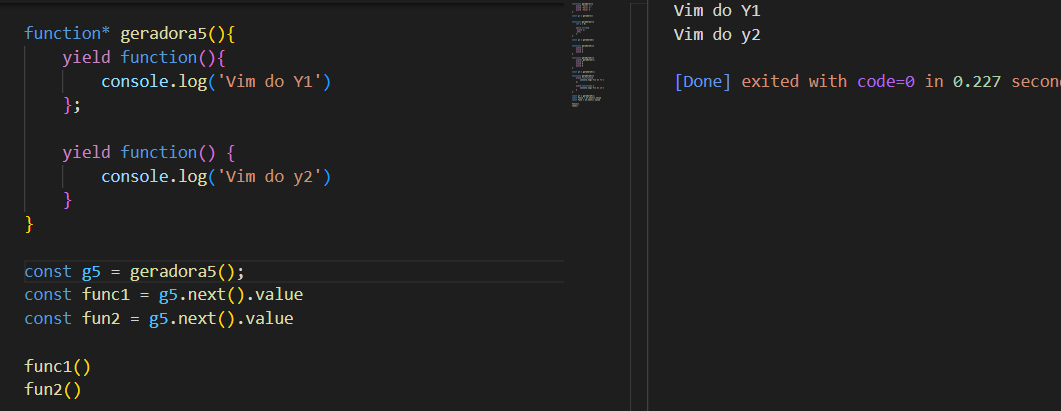
* Podemos utilizar essa função como parâmetros para outros comandos. Exemplo: Podemos utilizar a estrutura for para nos mostrar os valores de uma função geradora:



* Podemos utilizar as variáveis geradoras para realizar parte de uma atividade de outra função. Exemplo: Nessas funções, a função geradora vai ficar responsável de gerar os números iniciais (1, 2 e 3) enquanto a geradora 4 vai continuar a sequencia dos números. Para acionar a função geradora3 dentro da geradora4, utilizamos o comando yield\* nomedageradora().



* Funções geradoras não se limitam apenas a números, podemos utilizar strings e até mesmo funções dentro da função geradora. Exemplo: Nessa função, nosso sistema precisa que seja executada duas funções, porém necessita de uma ordem de execução



* Se utilizamos o comando return dentro de uma função geradora, ele encerra a função, então tudo que esteja após o comando, mesmo sendo um yield, não será executada