**JavaScript Funcional**

**Leonardo Moura Leitão (COD3R)**

Site: https://www.cod3r.com.br/

Código GitHub: https://github.com/cod3rcursos/javascript-funcional

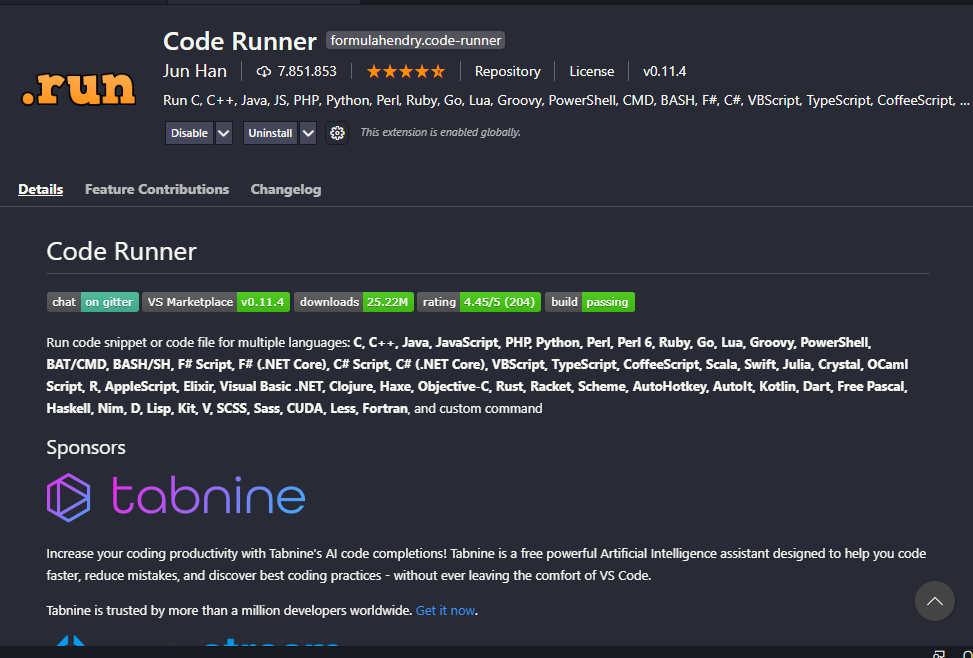
Vídeo: https://www.youtube.com/watch?v=W3f6hiTLipc

ID do certificado: rt1lrc9vcj

# Aula 01 - Fundamentos - Explicações iniciais

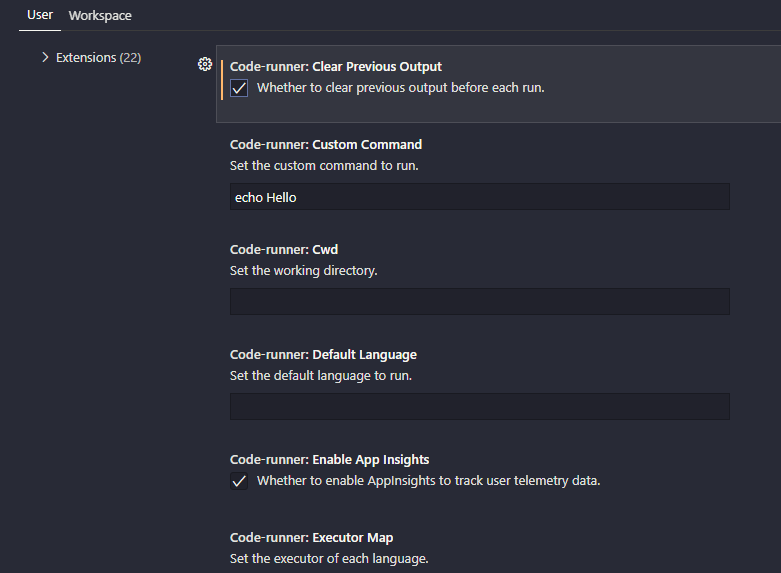
- Instale o Node e o Visual Studio Code.

- No Visual Studio Code instale o plugin "Code Runner":

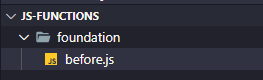


Ctrl + Alt + n (run code)

Ctrl + Alt + m (stop the running code)

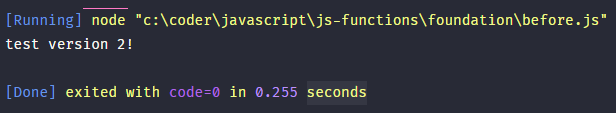


- No terminal, na página raiz do projeto (C:\coder\javascript\js-functions), adicione uma pasta chamada "foundation" e dentro dela insira um arquivo chamado "before.js":

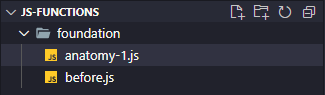


**foundation\before.js**

console.log('test version 2!')



# Aula 02 - Anatomia de uma função - Function declaration



**foundation\anatomy-1.js**

// Function declaration

// A função não recebe parâmetro e não retorna nada

function sayHello(){

console.log('Hello!')

}

sayHello()

// A função recebe parâmetro e não retorna nada

function sayHelloTo(name){

console.log('Hello ' + name)

console.log(`Hello ${name}`)

}

sayHelloTo('Mike')

// A função não recebe parâmetro e retorna o valor

function returnHi(){

return 'Hi!'

}

const greeting = returnHi()

console.log(greeting)

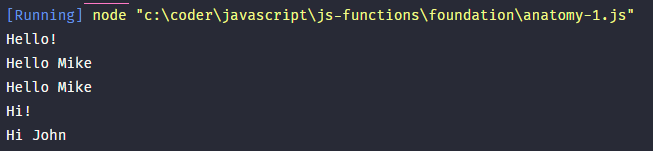
// A função recebe parâmetro e retorna o valor

function returnHiTo(name){

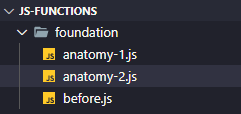
return `Hi ${name}`

}

console.log(returnHiTo('John'))



# Aula 03 - Anatomia de uma função - Function expression



É possível atribuir o valor de uma função a uma variável ou a uma constante. Isso se chama function expression.

**foundation\anatomy-2.js**

// Anonymous function

(function(a, b, c) {

return a + b + c

})

// Function expression

const sum = function (a, b) {

return a + b

}

const result = sum(7, 59)

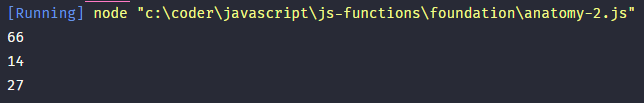
console.log(result)

const anotherSum = sum

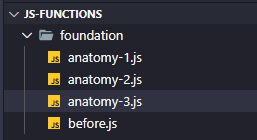
console.log(anotherSum(5, 9))

x = sum

console.log(x(11, 16))



# Aula 04 - Anatomia de uma função - Arrow Function



**foundation\anatomy-3.js**

// Function expression

const increment1 = function(n){

return n + 1

}

// Arrow Function is always anonymous

const increment2 = (n) => {

return n + 1

}

const increment3 = n => {

return n + 1

}

const increment4 = n => n + 1

const sum = (a, b) => a + b

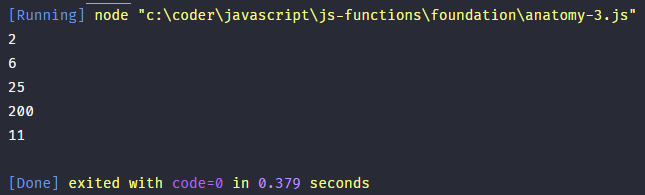
console.log(increment1(1))

console.log(increment2(5))

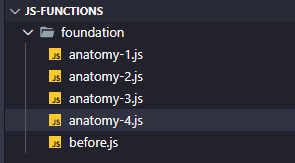
console.log(increment3(24))

console.log(increment4(199))

console.log(sum(3, 8))



# Aula 05 - Anatomia de uma função - Função imediatamente invocada



- Neste tipo de função use sempre ; no final

**foundation\anatomy-4.js**

// Anonymous Function

// IIFE - Immediately Invoked Function Expression

(function(a, b, c) {

let x = 3

console.log(`Result: ${a + b + c}`)

console.log(x)

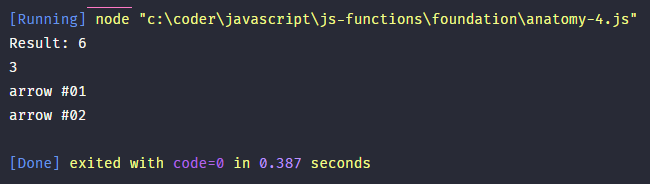
}) (1, 2, 3);

(() => {

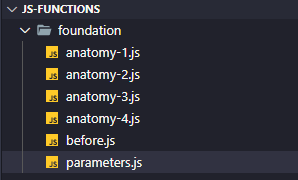
console.log('arrow #01')

})();

(() => console.log('arrow #02'))();



# Aula 06 - Parâmetros



**foundation\parameters.js**

function logParams(a, b, c) {

console.log(a, b, c)

}

logParams(1, 2, 3)

logParams(1, 2, 3, 4, 5)

logParams(1, 2)

logParams()

function defaultParams(a, b, c = 0) {

console.log(a, b, c)

}

defaultParams(7, 8, 9)

defaultParams(7, 8)

// Passando um array como parâmetro

function logNums(nums){

for(let n of nums){

console.log(n)

}

}

logNums([1, 2, 3])

// Operador spread/rest

function logNums2(...nums){

console.log(Array.isArray(nums))

console.log(nums)

for(let n of nums){

console.log(n)

}

}

logNums2(1, 2, 3)

function sumAll(...nums){

let total = 0

for(let n of nums){

total += n

}

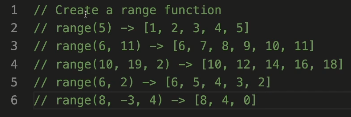
return total

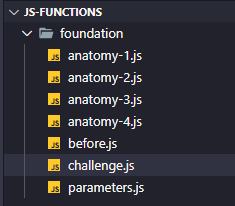
}

console.log(sumAll(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))



# Aulas 07 e 08 - Desafio 1





**foundation\challenge.js**

// Create a range function

function range (a, b, s = 1) {

const n1 = b === undefined ? 1 : a

const n2 = b === undefined ? a : b

const step = n1 <= n2 ? Math.abs(s) : -Math.abs(s)

const nums = []

for(let i = n1; n1 <= n2 ? i <= n2 : i >= n2; i += step) {

nums.push(i)

}

return nums

}

// range(5) -> [1, 2, 3, 4, 5]

console.log(range(5))

// range(6, 11) -> [6, 7, 8, 9, 10, 11]

console.log(range(6, 11))

// range(10, 19, 2) -> [10, 12, 14, 16, 18]

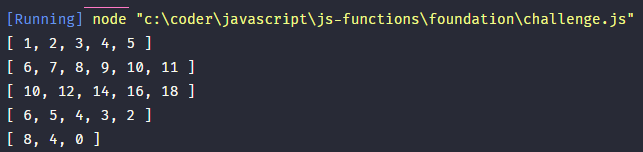
console.log(range(10, 19, 2))

// range(6, 2) -> [6, 5, 4, 3, 2]

console.log(range(6, 2))

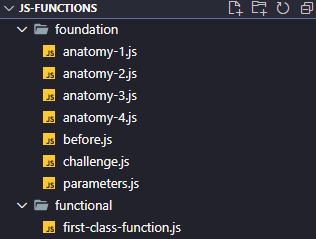
// range(8, -3, 4) -> [8, 4, 0]

console.log(range(8, -3, 4))



# Aula 09 - First-Class Function

- Dentro da pasta raiz do projeto, adicione uma subpasta chamada "functional" e dentro dela adicione um arquivo chamado "first-class-function.js":



**functional\first-class-function.js**

/\*

\* A programming language is said to have

\* First-class functions when functions in

\* that language are treated like any other

\* variable.

\*/

const add = function(a, b){

return a + b

}

const subtract = function(a, b){

return a - b

}

const multiply = (x, y) => x \* y

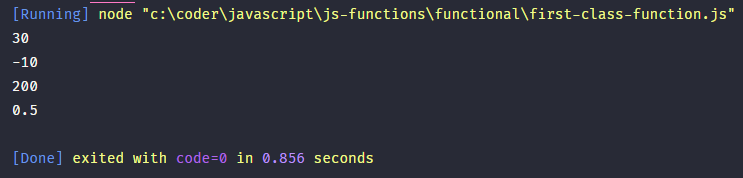
const divide = (x, y) => x / y

console.log(add(10, 20))

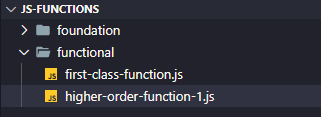
console.log(subtract(10, 20))

console.log(multiply(10, 20))

console.log(divide(10, 20))



# Aula 10 - Higher-Order Function - Parte 1



## Como passar uma função como parâmetro para outra função

**functional\higher-order-function-1.js**

/\*

\* Functions that operate on other functions,

\* either by taking them as arguments or by

\* returning them, are called higher-order functions.

\*/

function run(fn) {

return `Result: ${fn()}`

}

function sayHello() {

console.log('Hello!')

}

run(sayHello)

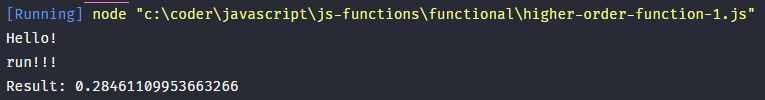
run(function(){

console.log('run!!!')

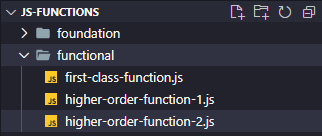
})

const result = run(Math.random)

console.log(result)



# Aula 11 - Higher-Order Function - Parte 2



## Como retornar uma função a partir de uma função

**functional\higher-order-function-2.js**

// curring

function finalPrice(tax) {

return function(price) {

return price \* (1 + tax)

}

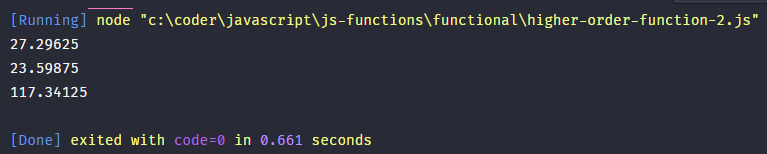
}

const nycFinalPrice = finalPrice(0.0875)

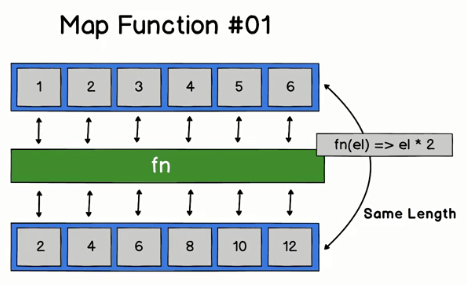
console.log(nycFinalPrice(25.1))

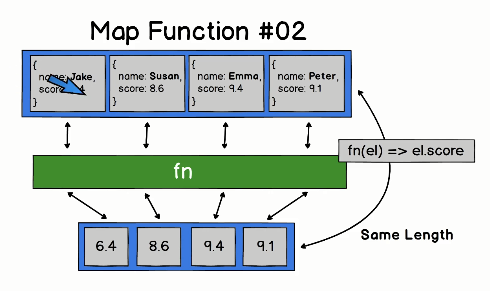
console.log(nycFinalPrice(21.7))

console.log(nycFinalPrice(107.9))

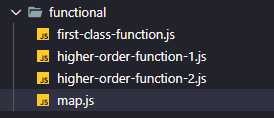


# Aula 12 - Map Function - Parte 1





# Aula 13 - Map Function - Parte 2



**functional\map.js**

const numbers = [1, 2, 3, 4, 5, 6]

// ================================================

const numberV2 = numbers.map(function(el) {

return el \* 2

})

console.log(numberV2)

// ================================================

const numberV3 = numbers.map((el) => {

return el \* 3

})

console.log(numberV3)

// ================================================

const numberV4 = numbers.map(el => el \* 4)

console.log(numberV4)

// ================================================

const numbersV5 = []

for(let n of numbers) {

numbersV5.push(n \* 2)

}

console.log(numbersV5)

// ================================================

const students = [

{ name: 'Jake', score: 6.4 },

{ name: 'Susan', score: 8.6 },

{ name: 'Emma', score: 9.4 },

{ name: 'Peter', score: 9.1 }

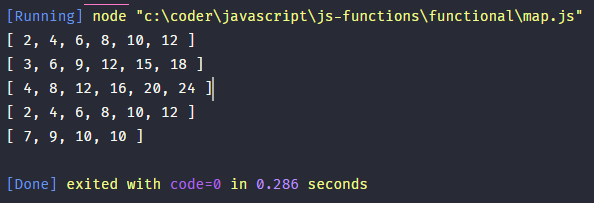
]

const getScore = el => el.score

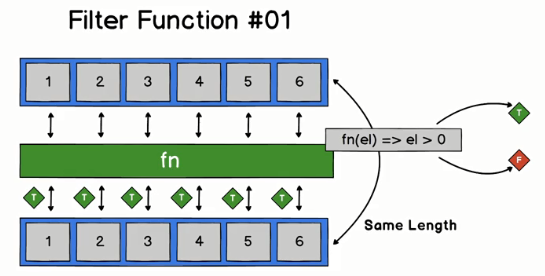
const result = students.map(getScore).map(Math.ceil)

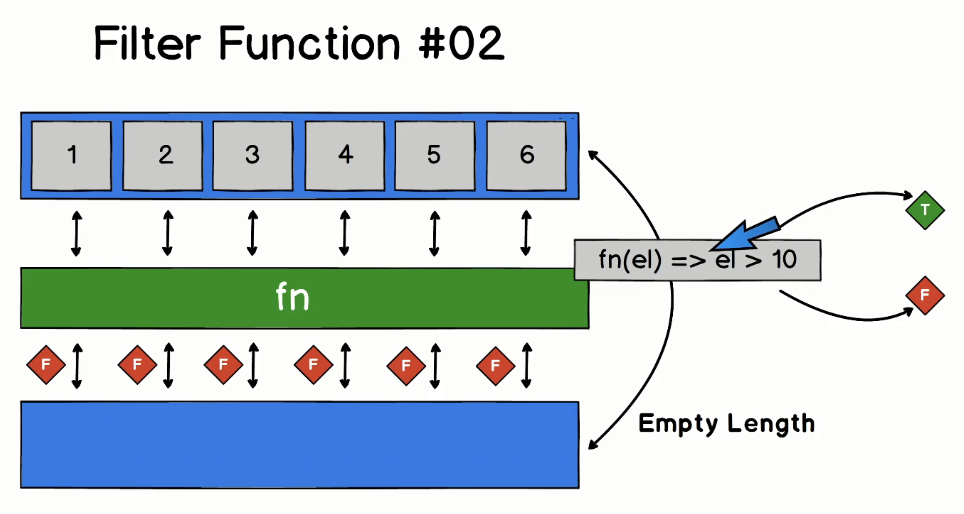
console.log(result)

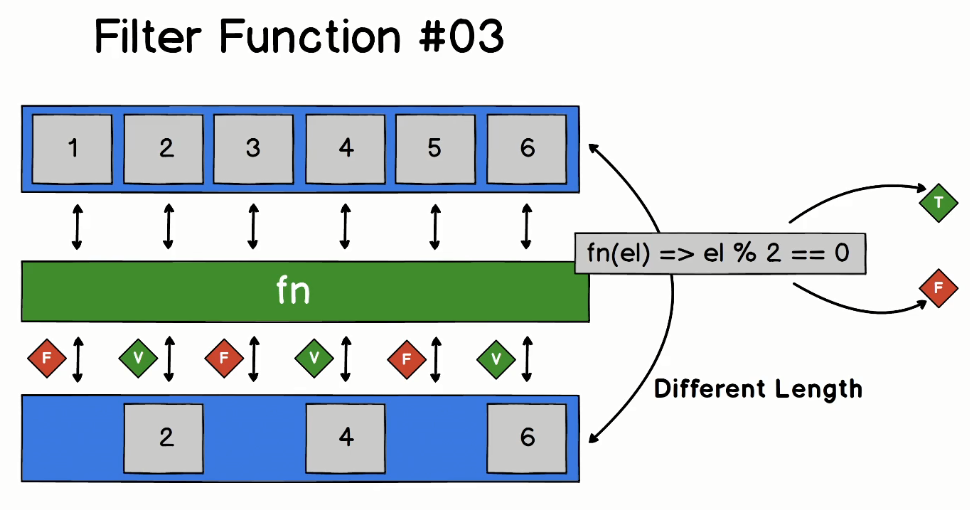
// ================================================

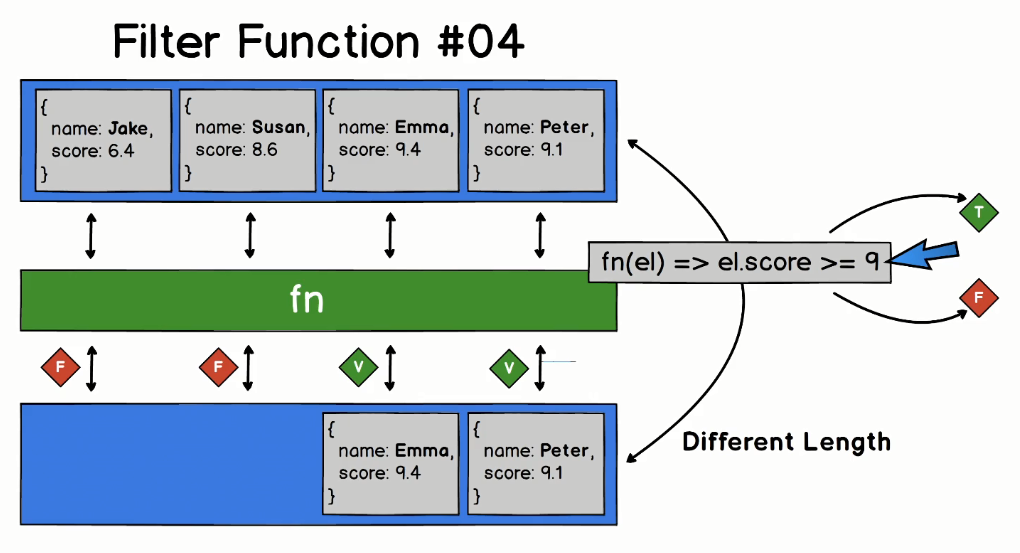


# Aula 14 - Filter Function - Parte 1

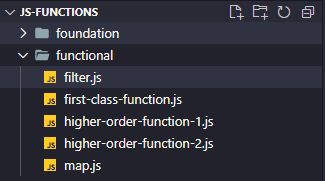








# Aula 15 - Filter Function - Parte 2



**functional\filter.js**

const numbers = [1, 2, 3, 4, 5, 6]

const greaterThanZero = el => el > 0

const greaterThanTen = el => el > 10

const even = el => el % 2 === 0

// console.log(numbers.filter(el => el > 0))

console.log(numbers.filter(greaterThanZero))

console.log(numbers.filter(greaterThanTen))

console.log(numbers.filter(even))

// =================================================

const students = [

{ name: 'Jake', score: 6.4 },

{ name: 'Susan', score: 8.6 },

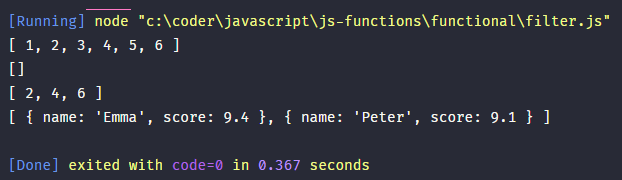
{ name: 'Emma', score: 9.4 },

{ name: 'Peter', score: 9.1 }

]

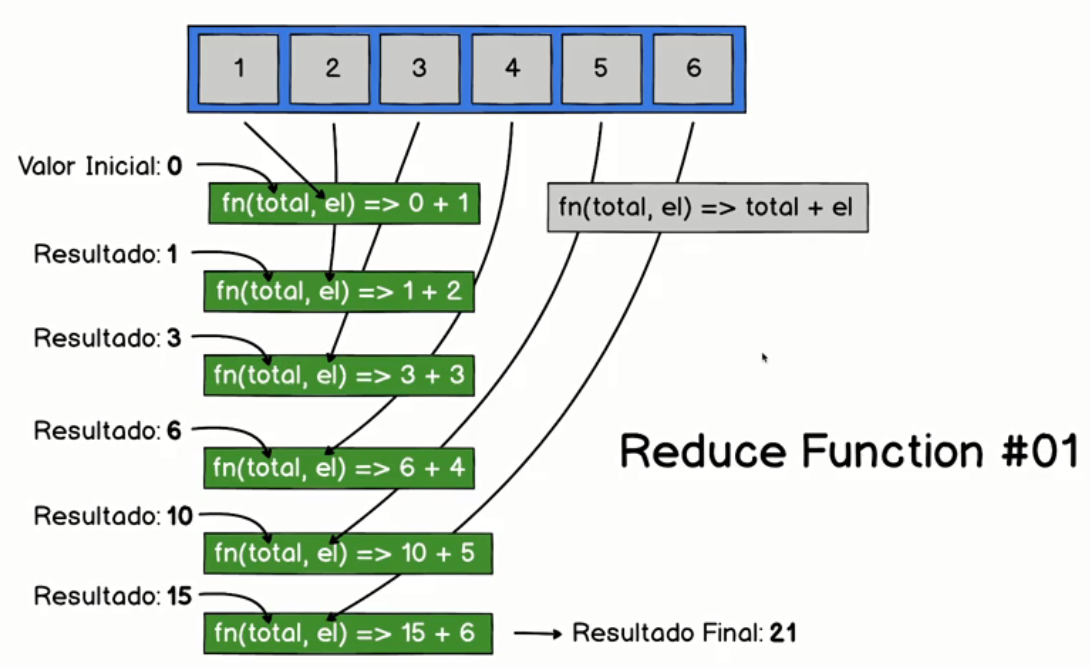
const greatStudent = student => student.score >= 9

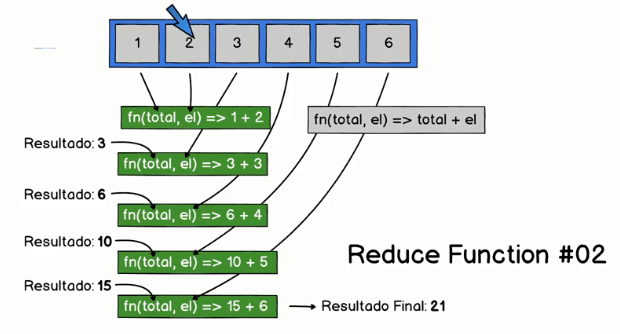
console.log(students.filter(greatStudent))



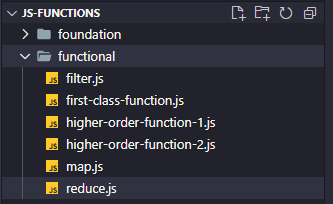
# Aula 16 - Reduce Function - Parte 1

## Somando todos os elementos do array





# Aula 17 - Reduce Function - Parte 2



**functional\reduce.js**

const numbers = [1, 2, 3, 4, 5, 6]

// Somando todos os elementos do array e adicionando um valor inicial

const sum = (total, el) => total + el

const total = numbers.reduce(sum, 100)

console.log(total)

// =====================================================================

// Calculando a média dos elementos de um array

const avg = (acc, el, i, array) => {

if(i === array.length - 1){

return (acc + el) / array.length

} else {

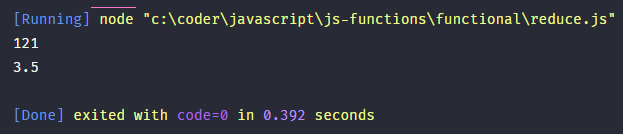
return acc + el

}

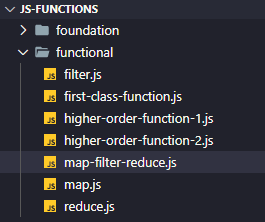
}

const result = numbers.reduce(avg)

console.log(result)



# Aula 18 - Map, Filter e Reduce



## Média dos alunos com nota maior ou igual a 9

**functional\map-filter-reduce.js**

const students = [

{ name: 'Jake', score: 6.4 },

{ name: 'Susan', score: 8.6 },

{ name: 'Emma', score: 9.4 },

{ name: 'Peter', score: 9.1 }

]

const greatStudent = student => student.score >= 9

const getScore = el => el.score

const avg = (acc, el, i, array) => {

if(i === array.length - 1){

return (acc + el) / array.length

} else {

return acc + el

}

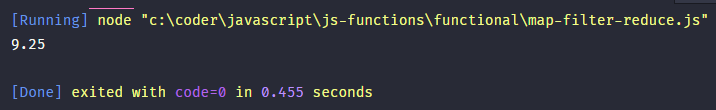
}

console.log(students

.filter(greatStudent)

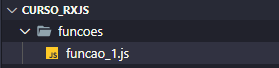
.map(getScore)

.reduce(avg))



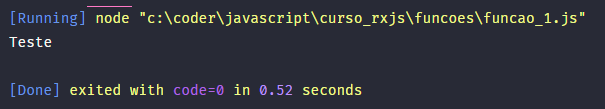
# Aula 19 - Funções - Configuração do Ambiente

- No terminal, na página raiz do projeto (C:\coder\javascript\curso\_rxjs), adicione uma pasta chamada "funcoes" e dentro dela insira um arquivo chamado "funcao\_1.js":



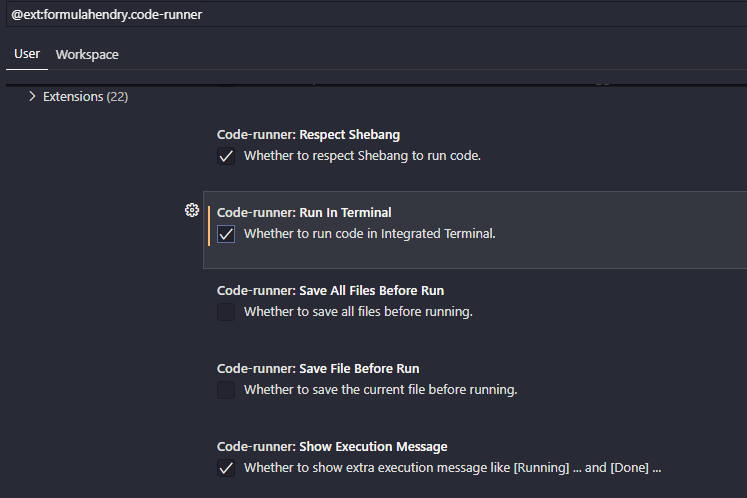
**funcoes\funcao\_1.js**

console.log('Teste')



## Executando o Code Runner no Terminal

- Marque a caixa "Code-runner: Run in Terminal"





# Aula 20 - Básico sobre função - Parte 1



**funcoes\basico\_1.js**

// Function Declaration

function bomDia() {

console.log('Bom dia!')

}

bomDia();

// Function expression

const boaTarde = function () {

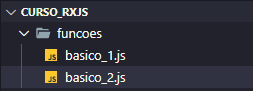
console.log('Boa tarde!')

}

boaTarde();



# Aula 21 - Básico sobre função - Parte 2



**funcoes\basico\_2.js**

function bomDia() {

console.log('Bom dia!')

}

const boaTarde = function () {

console.log('Boa tarde!')

}

// Passando uma função como parâmetro para outra função

function runFunction(fn){

if(typeof fn === 'function'){

fn()

}

}

runFunction(3)

runFunction(bomDia)

runFunction(boaTarde)

// Retornar uma função a partir de outra função

function potencia(base) {

return function(exp) {

return Math.pow(base, exp)

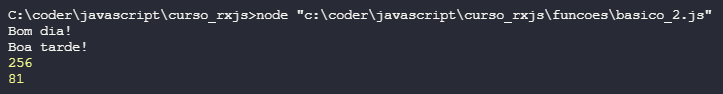
}

}

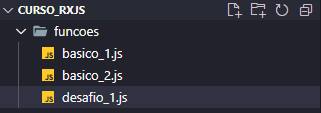
const potenciaDe2 = potencia(2)

console.log(potenciaDe2(8))

console.log(potencia(3)(4))



# Aula 22 - Funções - Desafio 1



**funcoes\desafio\_1.js**

// somar(3)(4)(5)

function somar(a) {

return function(b) {

return function(c) {

return a + b + c

}

}

}

console.log(somar(3)(4)(5))

const somarAB = somar(3)(4)

console.log(somarAB(13))

console.log(somar(13)(20)(30))

// =====================================================

// fn -> 3 \* 7

// fn -> 3 + 7

// fn -> 3 - 7

// calcular(3)(7)(fn)

function calcular(x){

return function(y){

return function (fn){

return fn(x, y)

}

}

}

function subtrair(a, b){

return a - b

}

function multiplicar(a, b){

return a \* b

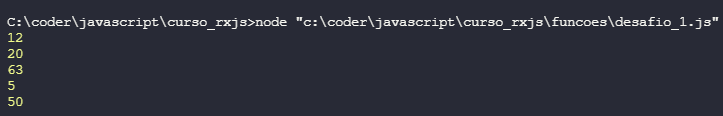
}

const r1 = calcular(10)(5)(subtrair)

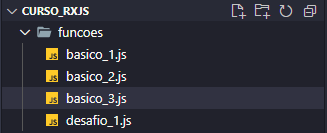
console.log(r1)

const r2 = calcular(10)(5)(multiplicar)

console.log(r2)



# Aula 23 - Básico sobre Função - Parte 3 (Arrow Function)



**funcoes\basico\_3.js**

// arrow function

const felizNatal = () => console.log('Feliz Natal!')

felizNatal()

// const saudacao = nome => "Fala " + nome + ", blz!?!"

const saudacao = nome => `Fala ${nome}, blz!?!`

console.log(saudacao('Maria'))

// =================================================

const somar = numeros => {

let total = 0

for(let n of numeros) {

total += n

}

return total

}

console.log(somar([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))

// ===================================================

const sum = (...num) => {

console.log(Array.isArray(num))

let total = 0

for(let n of num) {

total += n

}

return total

}

console.log(sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11))

// ===================================================

const potencia = (base) => {

return (exp) => {

return Math.pow(base, exp)

}

}

const pow = base => exp => Math.pow(base, exp)

console.log(potencia(2)(8))

console.log(potencia(3)(5))

// =====================================================

// this

Array.prototype.ultimo = function() {

console.log(this[this.length - 1])

}

Array.prototype.primeiro = function() {

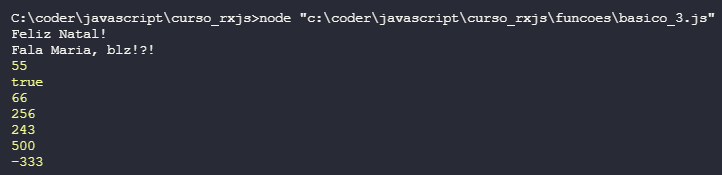
console.log(this[0])

}

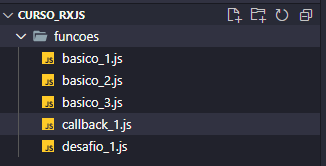
const numeros = [-333, 1, 2, 3, 500]

numeros.ultimo()

numeros.primeiro()



# Aula 24 - Função Callback - Parte 1



**funcoes\callback\_1.js**

function exec(fn, a, b) {

return fn(a, b)

}

const somarNoTerminal = (x, y) => console.log(x + y)

const subtrairNoTerminal = (w, z) => console.log(w - z)

exec(somarNoTerminal, 56, 38)

exec(subtrairNoTerminal, 182, 27)

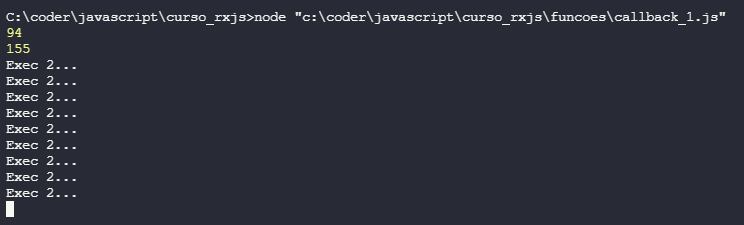
// const cb = () => console.log('Exec...')

// setInterval(cb, 1000)

setInterval(function () {

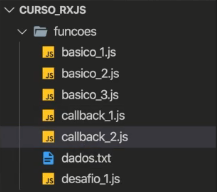
console.log('Exec 2...')

}, 1000)



# Aula 25 - Função Callback - Parte 2

- Adicione a pasta inicial do projeto dois arquivos: callback\_2.js e dados.txt



**funcoes\callback\_2.js**

const fs = require('fs')

const path = require('path')

const caminho = path.join(\_\_dirname, 'dados.txt')

function exibirConteudo(\_, conteudo) {

console.log(conteudo.toString())

}

// fs.readFile(caminho, {}, exibirConteudo)

console.log('Início...')

fs.readFile(caminho, exibirConteudo)

fs.readFile(caminho, (\_, conteudo) => console.log(conteudo.toString()))

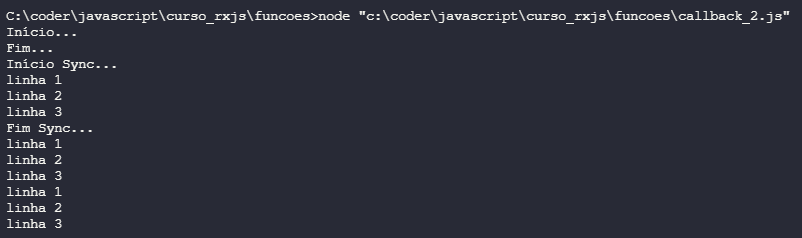
console.log('Fim...')

console.log('Início Sync...')

const conteudo = fs.readFileSync(caminho)

console.log(conteudo.toString())

console.log('Fim Sync...')



# Aula 26 - Função Map - Parte 1



[...].map(fn)

**funcoes\map\_1.js**

const result = [1, 2, 3, 4, 5, 6].map(el => el \* 2)

console.log(result);



# Aula 27 - Função Map - Parte 2

**funcoes\map\_2.js**

const nums = [1, 2, 3, 4, 5, 6]

const dobro = (n, i) => "posição " + i + " => " + (n \* 2)

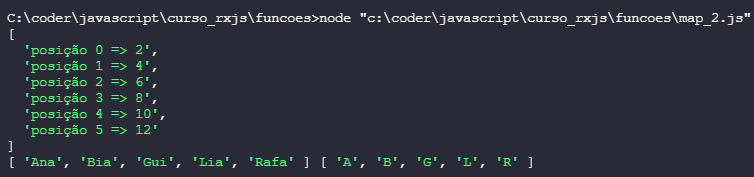
console.log(nums.map(dobro))

const nomes = ['Ana', 'Bia', 'Gui', 'Lia', 'Rafa']

const primeiraLetra = texto => texto[0]

const letras = nomes.map(primeiraLetra)

console.log(nomes, letras)



# Aula 28 - Função Map - Parte 3

**funcoes\map\_3.js**

const carrinho = [

{ nome: 'Caneta', qtde: 10, preco: 7.99 },

{ nome: 'Impressora', qtde: 0, preco: 649.50 },

{ nome: 'Caderno', qtde: 4, preco: 27.10 },

{ nome: 'Lápis', qtde: 3, preco: 5.82 },

{ nome: 'Tesoura', qtde: 1, preco: 19.20 }

]

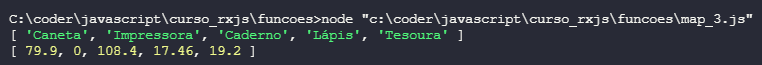
const getNome = item => item.nome

console.log(carrinho.map(getNome))

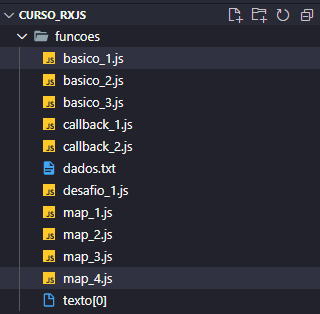
const getTotal = item => item.qtde \* item.preco

const totais = carrinho.map(getTotal)

console.log(totais)



# Aula 29 - Função Map - Parte 4



## Construindo o próprio map

**funcoes\map\_4.js**

const carrinho = [

{ nome: 'Caneta', qtde: 10, preco: 7.99 },

{ nome: 'Impressora', qtde: 0, preco: 649.50 },

{ nome: 'Caderno', qtde: 4, preco: 27.10 },

{ nome: 'Lápis', qtde: 3, preco: 5.82 },

{ nome: 'Tesoura', qtde: 1, preco: 19.20 }

]

Array.prototype.meuMap = function(fn) {

const novoArray = []

for(let i = 0; i < this.length; i++) {

novoArray.push(fn(this[i], i, this))

}

return novoArray

}

const getNome = item => item.nome

console.log(carrinho.meuMap(getNome))

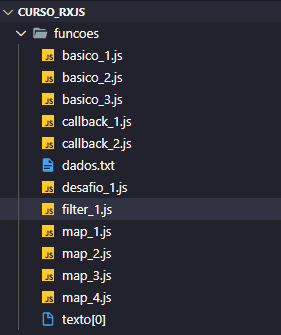
const getTotal = item => item.qtde \* item.preco

const totais = carrinho.meuMap(getTotal)

console.log(totais)



# Aula 30 - Função Filter - Parte 1



array.filter(fn)

**funcoes\filter\_1.js**

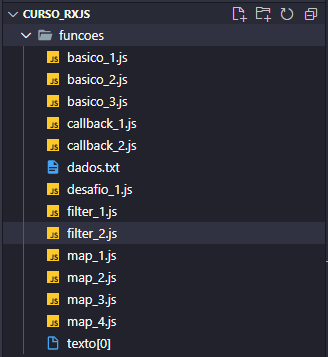
notas = [6, 5, 7, 3, 9, 10]

const aprovados = notas.filter(nota => nota >= 7)

console.log(aprovados)



# Aula 31 - Função Filter - Parte 2



**funcoes\filter\_2.js**

const carrinho = [

{ nome: 'Caneta', qtde: 10, preco: 7.99 },

{ nome: 'Impressora', qtde: 0, preco: 649.50 },

{ nome: 'Caderno', qtde: 4, preco: 27.10 },

{ nome: 'Lápis', qtde: 3, preco: 5.82 },

{ nome: 'Tesoura', qtde: 1, preco: 19.20 }

]

const getNome = item => item.nome

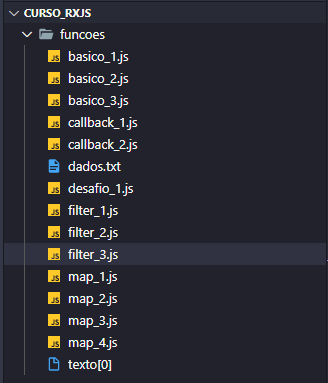
const qtdeMaiorQueZero = item => item.qtde > 0

const itensValidos = carrinho.filter(qtdeMaiorQueZero).map(getNome)

console.log(itensValidos)



# Aula 32 - Função Filter - Parte 3



**funcoes\filter\_3.js**

const carrinho = [

{ nome: 'Caneta', qtde: 10, preco: 7.99 },

{ nome: 'Impressora', qtde: 0, preco: 649.50 },

{ nome: 'Caderno', qtde: 4, preco: 27.10 },

{ nome: 'Lápis', qtde: 3, preco: 5.82 },

{ nome: 'Tesoura', qtde: 1, preco: 19.20 }

]

Array.prototype.meuFilter = function(fn) {

const novoArray = []

for(let i = 0; i < this.length; i++) {

if(fn(this[i], i, this)){

novoArray.push(this[i])

}

}

return novoArray

}

const getNome = item => item.nome

const qtdeMaiorQueZero = item => item.qtde > 0

const itensValidos = carrinho.meuFilter(qtdeMaiorQueZero).map(getNome)

console.log(itensValidos)



# Aula 33 - Função Reduce - Parte 1

array.reduce(fn, i)

fn = função

i = valor inicial

O resultado final de um reduce é um número.

## Somar todos os números de um array

**funcoes\reduce\_1.js**

nums = [3, 2, 1, -3, 4, 7]

const soma = (acc, el) => acc + el

console.log(nums.reduce(soma))



# Aula 34 - Função Reduce - Parte 2

**funcoes\reduce\_2.js**

const carrinho = [

{ nome: 'Caneta', qtde: 10, preco: 7.99 },

{ nome: 'Impressora', qtde: 0, preco: 649.50 },

{ nome: 'Caderno', qtde: 4, preco: 27.10 },

{ nome: 'Lápis', qtde: 3, preco: 5.82 },

{ nome: 'Tesoura', qtde: 1, preco: 19.20 }

]

const totais = item => item.qtde \* item.preco

const somar = (acc, el) => acc + el

const totalGeral = carrinho.map(totais).reduce(somar)

console.log(totalGeral)



# Aulas 35 e 36 - Desafio Map, Filter e Reduce

**funcoes\desafio\_2.js**

const carrinho = [

{ nome: 'Caneta', qtde: 10, preco: 7.99, fragil: true },

{ nome: 'Impressora', qtde: 1, preco: 649.50, fragil: true },

{ nome: 'Caderno', qtde: 4, preco: 27.10, fragil: false },

{ nome: 'Lápis', qtde: 3, preco: 5.82, fragil: false },

{ nome: 'Tesoura', qtde: 1, preco: 19.20, fragil: true }

]

// filter, map, reduce

// 1. fragil: true

// 2. qtde \* preco -> total

// 3. media totais

const fragil = item => item.fragil

const getTotal = item => item.qtde \* item.preco

const getMedia = (acc, el) => {

const novaQtde = acc.qtde + 1

const novoTotal = acc.total + el

return {

qtde: novaQtde,

total: novoTotal,

media: novoTotal / novaQtde

}

}

const mediaInicial = { qtde: 0, total:0, media:0 }

const media = carrinho

.filter(fragil)

.map(getTotal)

.reduce(getMedia, mediaInicial)

.media

console.log(`A média é ${media}!`)



# Aula 37 - Função Reduce - Parte 3

**funcoes\reduce\_3.js**

nums = [3, 2, 1, -3, 4, 7]

Array.prototype.meuReduce = function(fn, inicial) {

let acc = inicial

for(let i = 0; i < this.length; i++) {

if(!acc && i === 0) {

acc = this[i]

} else {

acc = fn(acc, this[i], i, this)

}

}

return acc

}

const soma = (acc, el) => acc + el

console.log(nums.meuReduce(soma))



# Aula 38 - Promise - Parte 1

**funcoes\promise\_1.js**

let p = new Promise(function(cumprirPromessa) {

cumprirPromessa({

x: 3,

y: 4

})

})

p.then(function(valor){

console.log(valor)

console.log(valor.x)

})

// ==============================================================

const primeiroElemento = array => array[0]

const primeiraLetra = string => string[0]

const letraMinuscula = letra => letra.toLowerCase()

new Promise(function(resolve){

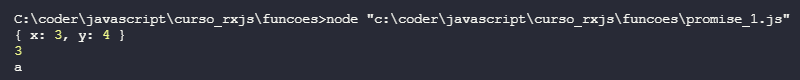
resolve(['Ana', 'Bia', 'Carlos', 'Daniel'])

}).then(primeiroElemento)

.then(primeiraLetra)

.then(letraMinuscula)

.then(console.log)



# Aula 39 - Promise - Parte 2

**funcoes\promise\_2a.js**

// callback hell

setTimeout(function() {

console.log('Executando callback...')

setTimeout(function() {

console.log('Executando callback...')

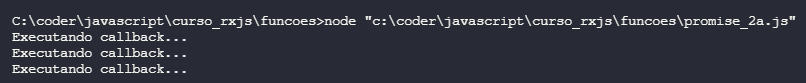
setTimeout(function() {

console.log('Executando callback...')

}, 2000)

}, 2000)

}, 2000)



**funcoes\promise\_2b.js**

function esperarPor(tempo = 2000) {

return new Promise(function(resolve){

setTimeout(function(){

console.log('Executando promise...')

resolve('Executado...')

}, tempo)

})

}

esperarPor(3000).then(texto => console.log(texto))



**funcoes\promise\_2c.js**

function esperarPor(tempo = 2000) {

return new Promise(function(resolve){

setTimeout(function(){

console.log('Executando promise...')

resolve()

}, tempo)

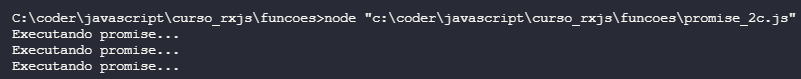
})

}

esperarPor()

.then(() => esperarPor())

.then(esperarPor)



# Aula 40 - Promise - Parte 3

**funcoes\promise\_3.js**

function gerarNumerosEntre(min, max) {

if(min > max) {

[max, min] = [min, max]

}

return new Promise(resolve => {

const fator = max - min + 1

const aleatorio = parseInt(Math.random() \* fator) + min

resolve(aleatorio)

})

}

gerarNumerosEntre(1, 60)

.then(num => num \* 10)

.then(numX10 => `O número gerado foi ${numX10}`)

.then(console.log)









# Aulas 41 e 42 - Desafio Promise

**funcoes\desafio\_3a.js**

const fs = require('fs')

const path = require('path')

function lerArquivo(caminho) {

return new Promise(resolve => {

fs.readFile(caminho, function(\_, conteudo){

resolve(conteudo.toString())

})

console.log('Depois de ler')

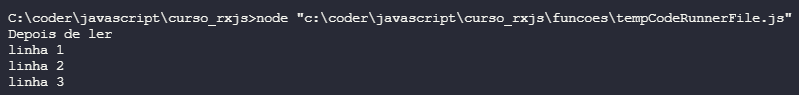
})

}

const caminho = path.join(\_\_dirname, 'dados.txt')

lerArquivo(caminho)

.then(conteudo => console.log(conteudo))



**funcoes\desafio\_3b.js**

const fs = require('fs')

const path = require('path')

function lerArquivo(caminho) {

return new Promise(resolve => {

fs.readFile(caminho, function(\_, conteudo){

resolve(conteudo.toString())

})

})

}

const caminho = path.join(\_\_dirname, 'dados.txt')

lerArquivo(caminho)

.then(conteudo => conteudo.split('\n'))

.then(linhas => console.log('Número de linhas: ' + linhas.length + ' - Segunda linha:', linhas[1]))



**funcoes\desafio\_3c.js**

const fs = require('fs')

const path = require('path')

function lerArquivo(caminho) {

return new Promise(resolve => {

fs.readFile(caminho, function(\_, conteudo) {

resolve(conteudo.toString())

})

})

}

const caminho = path.join(\_\_dirname, 'dados.txt')

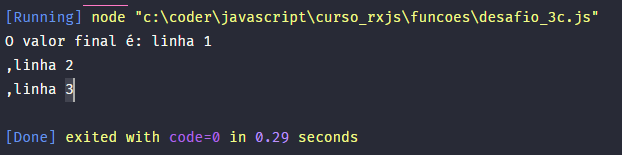
lerArquivo(caminho)

.then(conteudo => conteudo.split('\n'))

.then(linhas => linhas.join(','))

.then(conteudo => `O valor final é: ${conteudo}`)

.then(console.log)



# Aula 43 - Promise - Parte 4

**funcoes\promise\_4a.js**

function gerarNumerosEntre(min, max, tempo) {

if(min > max) {

[max, min] = [min, max]

}

return new Promise(resolve => {

setTimeout(function() {

const fator = max - min + 1

const aleatorio = parseInt(Math.random() \* fator) + min

resolve(aleatorio)

}, tempo)

})

}

function gerarVariosNumeros() {

return Promise.all([

gerarNumerosEntre(1, 60, 4000),

gerarNumerosEntre(1, 60, 1000),

gerarNumerosEntre(1, 60, 500),

gerarNumerosEntre(1, 60, 3000),

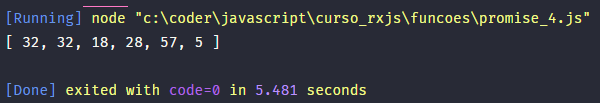
gerarNumerosEntre(1, 60, 100),

gerarNumerosEntre(1, 60, 1500)

])

}

gerarVariosNumeros().then(numeros => console.log(numeros))



**funcoes\promise\_4b.js**

function gerarNumerosEntre(min, max, tempo) {

if(min > max) {

[max, min] = [min, max]

}

return new Promise(resolve => {

setTimeout(function() {

const fator = max - min + 1

const aleatorio = parseInt(Math.random() \* fator) + min

resolve(aleatorio)

}, tempo)

})

}

function gerarVariosNumeros() {

return Promise.all([

gerarNumerosEntre(1, 60, 4000),

gerarNumerosEntre(1, 60, 1000),

gerarNumerosEntre(1, 60, 500),

gerarNumerosEntre(1, 60, 3000),

gerarNumerosEntre(1, 60, 100),

gerarNumerosEntre(1, 60, 1500)

])

}

console.time('promise')

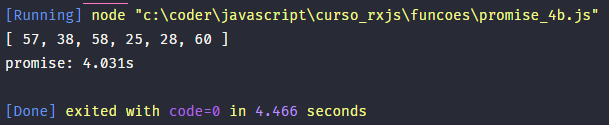
gerarVariosNumeros()

.then(console.log)

.then(() => {

console.timeEnd('promise')

})



# Aula 44 - Promise - Parte 5

**funcoes\promise\_5a.js**

function funcionarOuNao(valor, chanceErro) {

return new Promise((resolve, reject) => {

try {

console.log('temp')

const valorSorteado = Math.random()

if(valorSorteado < chanceErro) {

console.log(`Valor sorteado: ${valorSorteado}`)

reject('Ocorreu um erro!')

} else {

console.log(`Valor sorteado: ${valorSorteado}`)

resolve(valor)

}

} catch(e) {

reject(e)

}

})

}

funcionarOuNao('Testando...', 0.5)

.then(v => `Valor: ${v}`)

.then(

v => consol.log(v),

err => console.log(`Erro Esp.: ${err}`)

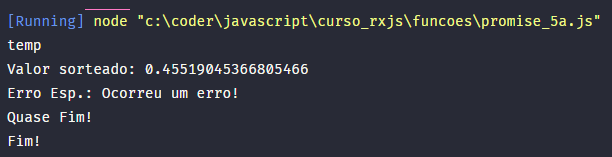
)

.then(() => console.log('Quase Fim!'))

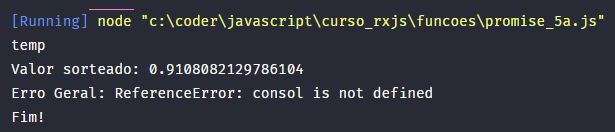
.catch(err => console.log(`Erro Geral: ${err}`))

.then(() => console.log('Fim!'))

- Se valorSorteado < 0.5:



- Se valorSorteado > 0.5:



**funcoes\promise\_5b.js**

function funcionarOuNao(valor, chanceErro) {

return new Promise((resolve, reject) => {

try {

con.log('temp')

const valorSorteado = Math.random()

if(valorSorteado < chanceErro) {

console.log(`Valor sorteado: ${valorSorteado}`)

reject('Ocorreu um erro!')

} else {

console.log(`Valor sorteado: ${valorSorteado}`)

resolve(valor)

}

} catch(e) {

reject(e)

}

})

}

funcionarOuNao('Testando...', 0.5)

.then(v => `Valor: ${v}`)

.then(

v => consol.log(v),

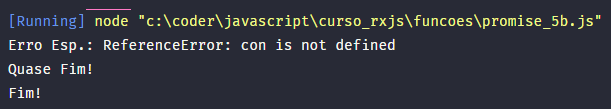
err => console.log(`Erro Esp.: ${err}`)

)

.then(() => console.log('Quase Fim!'))

.catch(err => console.log(`Erro Geral: ${err}`))

.then(() => console.log('Fim!'))



# Aula 45 - Async/Await - Parte 1

**funcoes\async\_await\_1a.js**

function esperarPor(tempo = 2000) {

return new Promise(function(resolve) {

setTimeout(() => resolve(), tempo)

})

}

esperarPor(2000)

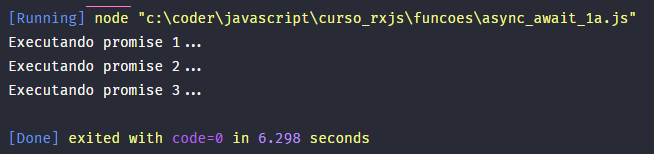
.then(() => console.log('Executando promise 1...'))

.then(esperarPor)

.then(() => console.log('Executando promise 2...'))

.then(esperarPor)

.then(() => console.log('Executando promise 3...'))



**funcoes\async\_await\_1b.js**

function esperarPor(tempo = 2000) {

return new Promise(function(resolve) {

setTimeout(() => resolve(), tempo)

})

}

async function executar() {

esperarPor(1500)

console.log('Async/Await 1...')

esperarPor(1500)

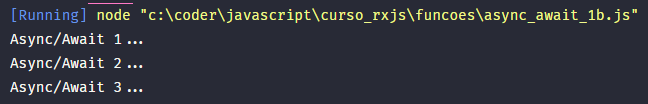
console.log('Async/Await 2...')

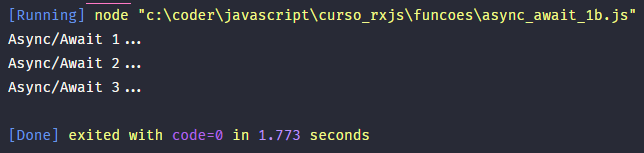
esperarPor(1500)

console.log('Async/Await 3...')

}

executar()





**funcoes\async\_await\_1c.js**

function esperarPor(tempo = 2000) {

return new Promise(function(resolve) {

setTimeout(() => resolve(), tempo)

})

}

async function executar() {

await esperarPor(1500)

console.log('Async/Await 1...')

await esperarPor(1500)

console.log('Async/Await 2...')

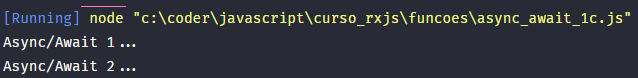
await esperarPor(1500)

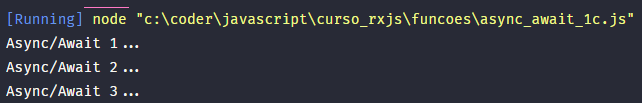
console.log('Async/Await 3...')

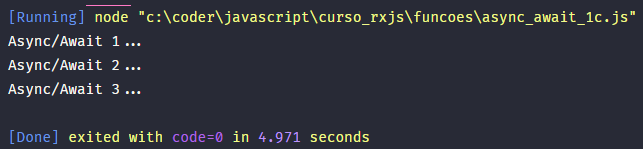
}

executar()









**funcoes\async\_await\_1d.js**

function esperarPor(tempo = 2000) {

return new Promise(function(resolve) {

setTimeout(() => resolve(), tempo)

})

}

function retornarValor() {

return new Promise(resolve => {

setTimeout(() => resolve(10), 5000)

})

}

async function executar() {

let valor = await retornarValor()

await esperarPor(1500)

console.log(`Async/Await ${valor}...`)

await esperarPor(1500)

console.log(`Async/Await ${valor + 1}...`)

await esperarPor(1500)

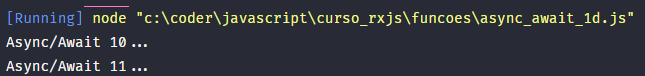
console.log(`Async/Await ${valor + 2}...`)

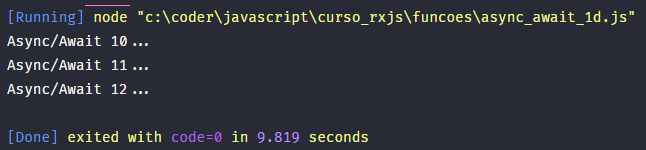
}

executar()









**funcoes\async\_await\_1e.js**

function esperarPor(tempo = 2000) {

return new Promise(function(resolve) {

setTimeout(() => resolve(), tempo)

})

}

function retornarValor() {

return new Promise(resolve => {

setTimeout(() => resolve(10), 5000)

})

}

async function executar() {

let valor = await retornarValor()

await esperarPor(1500)

console.log(`Async/Await ${valor}...`)

await esperarPor(1500)

console.log(`Async/Await ${valor + 1}...`)

await esperarPor(1500)

console.log(`Async/Await ${valor + 2}...`)

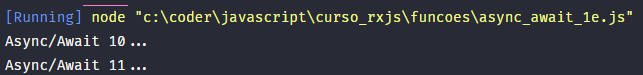
return valor + 3

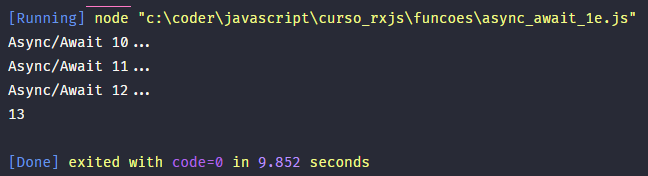
}

executar().then(console.log)









**funcoes\async\_await\_1f.js**

function esperarPor(tempo = 2000) {

return new Promise(function(resolve) {

setTimeout(() => resolve(), tempo)

})

}

function retornarValor() {

return new Promise(resolve => {

setTimeout(() => resolve(10), 5000)

})

}

async function executar() {

let valor = await retornarValor()

await esperarPor(1500)

console.log(`Async/Await ${valor}...`)

await esperarPor(1500)

console.log(`Async/Await ${valor + 1}...`)

await esperarPor(1500)

console.log(`Async/Await ${valor + 2}...`)

return valor + 3

}

async function executarDeVerdade() {

const valor = await executar()

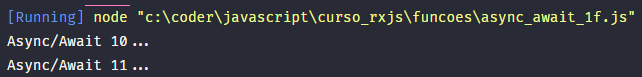
console.log(valor)

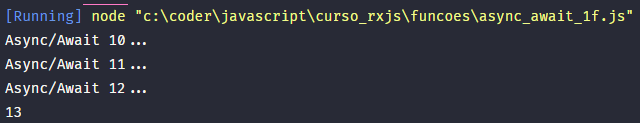
}

executarDeVerdade()









**funcoes\async\_await\_1g.js**

function esperarPor(tempo = 2000) {

return new Promise(function(resolve) {

setTimeout(() => resolve(), tempo)

})

}

function retornarValor() {

return new Promise(resolve => {

setTimeout(() => resolve(10), 5000)

})

}

async function retornarValorRapido() {

return 20

}

async function executar() {

let valor = await retornarValorRapido()

await esperarPor(1500)

console.log(`Async/Await ${valor}...`)

await esperarPor(1500)

console.log(`Async/Await ${valor + 1}...`)

await esperarPor(1500)

console.log(`Async/Await ${valor + 2}...`)

return valor + 3

}

async function executarDeVerdade() {

const valor = await executar()

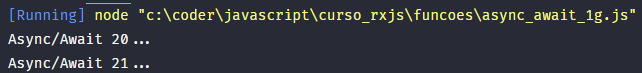
console.log(valor)

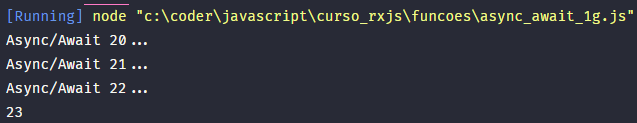
}

executarDeVerdade()









# Aula 46 - Async/Await - Parte 2

**funcoes\async\_await\_2a.js**

function gerarNumerosEntre(min, max, numerosProibidos) {

if(min > max) [max, min] = [min, max]

return new Promise((resolve, reject) => {

const fator = max - min + 1

const aleatorio = parseInt(Math.random() \* fator) + min

console.log(aleatorio)

if(numerosProibidos.includes(aleatorio)) {

reject('Número repetido!')

} else {

resolve(aleatorio)

}

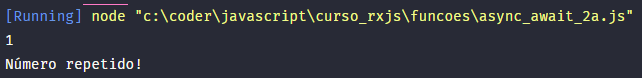
})

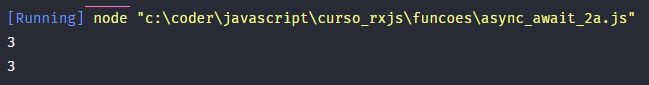
}

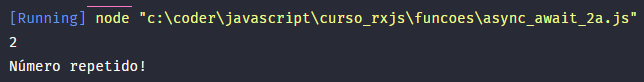
gerarNumerosEntre(1, 5, [1, 2, 4])

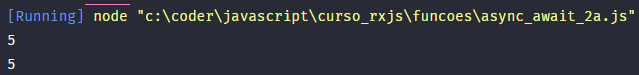
.then(console.log)

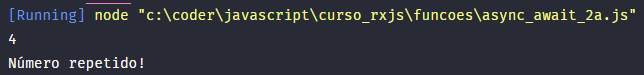
.catch(console.log)











**funcoes\async\_await\_2b.js**

function gerarNumerosEntre(min, max, numerosProibidos) {

if(min > max) [max, min] = [min, max]

return new Promise((resolve, reject) => {

const fator = max - min + 1

const aleatorio = parseInt(Math.random() \* fator) + min

if(numerosProibidos.includes(aleatorio)) {

reject('Número repetido!')

} else {

resolve(aleatorio)

}

})

}

async function gerarMegoSena(qtdeNumeros) {

const numeros = []

for(let \_ of Array(qtdeNumeros).fill()) {

numeros.push(await gerarNumerosEntre(1, 60, numeros))

}

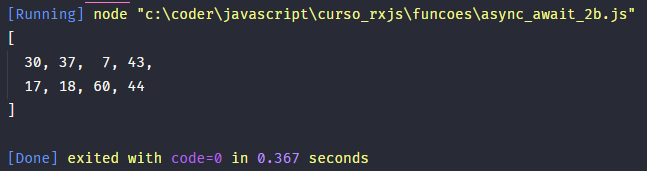
return numeros

}

gerarMegoSena(8)

.then(console.log)

.catch(console.log)



**funcoes\async\_await\_2c.js**

function gerarNumerosEntre(min, max, numerosProibidos) {

if(min > max) [max, min] = [min, max]

return new Promise((resolve, reject) => {

const fator = max - min + 1

const aleatorio = parseInt(Math.random() \* fator) + min

if(numerosProibidos.includes(aleatorio)) {

reject('Número repetido!')

} else {

resolve(aleatorio)

}

})

}

async function gerarMegaSena(qtdeNumeros) {

try {

const numeros = []

for(let \_ of Array(qtdeNumeros).fill()) {

numeros.push(await gerarNumerosEntre(1, 60, numeros))

}

return numeros

} catch (e) {

throw "Que chato!!!"

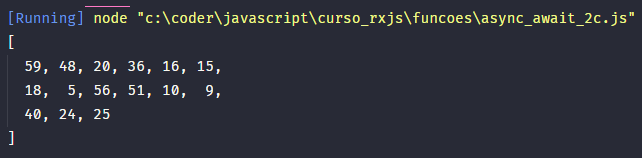
}

}

gerarMegaSena(15)

.then(console.log)

.catch(console.log)





**funcoes\async\_await\_2d.js**

function gerarNumerosEntre(min, max, numerosProibidos) {

if(min > max) [max, min] = [min, max]

return new Promise((resolve, reject) => {

const fator = max - min + 1

const aleatorio = parseInt(Math.random() \* fator) + min

if(numerosProibidos.includes(aleatorio)) {

reject('Número repetido!')

} else {

resolve(aleatorio)

}

})

}

async function gerarMegaSena(qtdeNumeros, tentativas = 1) {

try {

const numeros = []

for(let \_ of Array(qtdeNumeros).fill()) {

numeros.push(await gerarNumerosEntre(1, 60, numeros))

}

return numeros

} catch(e) {

if(tentativas > 100) {

throw "Não deu certo!"

} else {

return gerarMegaSena(qtdeNumeros, tentativas + 1)

}

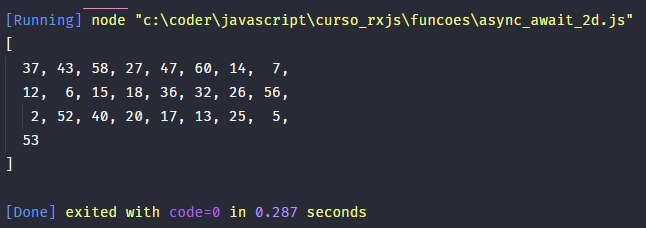
}

}

gerarMegaSena(25)

.then(console.log)

.catch(console.log)





# Aula 47 - OO - Parte 1

**funcoes\oo\_1a.js**

function Produto(nome, preco) {

this.nome = nome

this.preco = preco

let privado = 3

}

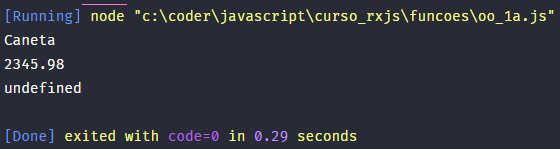
const p1 = new Produto('Caneta', 8.59)

console.log(p1.nome)

const p2 = new Produto('Geladeira', 2345.98)

console.log(p2.preco)

console.log(p2.privado)



**funcoes\oo\_1b.js**

// Função Contrutora

function Produto(nome, preco, desc = 0.50) {

this.nome = nome

this.preco = preco

this.desc = desc

this.precoFinal = function() {

return this.preco \* (1 - this.desc)

}

}

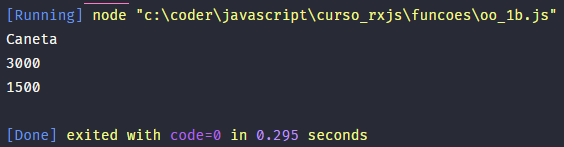
const p1 = new Produto('Caneta', 10)

console.log(p1.nome)

const p2 = new Produto('Geladeira', 3000)

console.log(p2.preco)

console.log(p2.precoFinal())



# Aula 48 - OO - Parte 2

**funcoes\oo\_2a.js**

class Produto {

constructor(nome, preco, desc = 0.5) {

this.nome = nome

this.preco = preco

this.desc = desc

}

precoFinal(){

return this.preco \* (1 - this.desc)

}

}

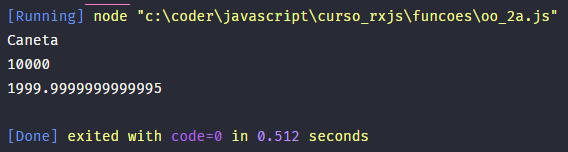
const p1 = new Produto('Caneta', 10)

console.log(p1.nome)

const p2 = new Produto('Geladeira', 10000, 0.8)

console.log(p2.preco)

console.log(p2.precoFinal())



**funcoes\oo\_2b.js**

class Produto {

constructor(nome, preco, desc = 0.5) {

this.\_nome = nome

this.preco = preco

this.desc = desc

}

get nome() {

return `Produto: ${this.\_nome}`

}

get precoFinal(){

return this.preco \* (1 - this.desc)

}

}

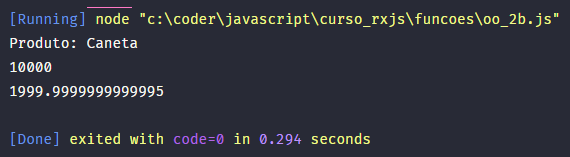
const p1 = new Produto('Caneta', 10)

console.log(p1.nome)

const p2 = new Produto('Geladeira', 10000, 0.8)

console.log(p2.preco)

console.log(p2.precoFinal)



**funcoes\oo\_2c.js**

class Produto {

constructor(nome, preco, desc = 0.5) {

this.\_nome = nome

this.preco = preco

this.desc = desc

}

get nome() {

return `Produto: ${this.\_nome}`

}

set nome(novoNome) {

this.\_nome = novoNome.toUpperCase()

}

get precoFinal(){

return this.preco \* (1 - this.desc)

}

}

const p1 = new Produto('Caneta', 10)

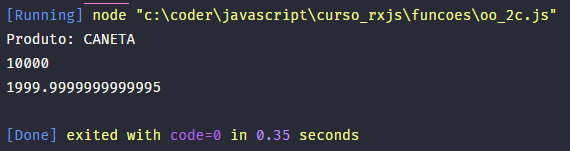
p1.nome = 'caneta'

console.log(p1.nome)

const p2 = new Produto('Geladeira', 10000, 0.8)

console.log(p2.preco)

console.log(p2.precoFinal)



**funcoes\oo\_2d.js**

class Produto {

constructor(nome, preco, desc = 0.5) {

this.nome = nome

this.preco = preco

this.desc = desc

}

get nome() {

return `Produto: ${this.\_nome}`

}

set nome(novoNome) {

this.\_nome = novoNome.toUpperCase()

}

get preco() {

return this.\_preco

}

set preco(novoPreco) {

if(novoPreco >= 0) {

this.\_preco = novoPreco

}

}

get precoFinal() {

return this.preco \* (1 - this.desc)

}

}

const p1 = new Produto('Caneta', 10)

// p1.nome = 'caneta'

p1.preco = -20

console.log(p1.nome)

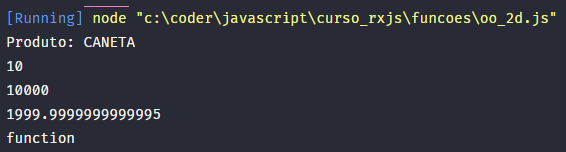
console.log(p1.preco)

const p2 = new Produto('Geladeira', 10000, 0.8)

console.log(p2.preco)

console.log(p2.precoFinal)

console.log(typeof Produto)



# Aula 49 - OO - Parte 3

**funcoes\oo\_3a.js**

// Função Contrutora

function Produto(nome, preco, desc = 0.50) {

this.nome = nome

this.preco = preco

this.\_desc = desc

this.precoFinal = function() {

return this.preco \* (1 - this.\_desc)

}

}

Produto.prototype.log = function() {

console.log(`Nome: ${this.nome} Preço: R$ ${this.preco}`)

}

const p1 = new Produto('Caneta', 10)

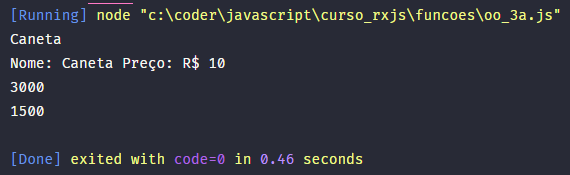
console.log(p1.nome)

p1.log()

const p2 = new Produto('Geladeira', 3000)

console.log(p2.preco)

console.log(p2.precoFinal())



**funcoes\oo\_3b.js**

// Função Contrutora

function Produto(nome, preco, desc = 0.50) {

this.nome = nome

this.preco = preco

this.\_desc = desc

this.precoFinal = function() {

return this.preco \* (1 - this.\_desc)

}

}

Produto.prototype.log = function() {

console.log(`Nome: ${this.nome} Preço: R$ ${this.preco}`)

}

Object.defineProperty(Produto.prototype, 'desc', {

get: function() {

return this.\_desc

},

set: function(novoDesc) {

if(novoDesc >= 0 && novoDesc <= 1) {

this.\_desc = novoDesc

}

}

})

Object.defineProperty(Produto.prototype, 'descString', {

get: function() {

return `${this.\_desc \* 100}% de desconto!`

},

})

const p1 = new Produto('Caneta', 10)

console.log(p1.nome)

p1.log()

const p2 = new Produto('Geladeira', 3000)

console.log(p2.preco)

console.log(p2.precoFinal())

p2.desc = 0.99

console.log(p2.desc)

console.log(p2.descString)

