Curso de Java Script – Roger Mello

# Aula 001

40:01

Aula inaugural

Visão do Roger sobre como realizar um curso, suas ideias e regras.

Comece e não pare, devagar ou rápido, não pare, não pule as aulas e não faça diferente do professor perdendo o foco

Entender é diferente de executar, pratique, mesmo que vc tenha entendido tudo

Nove marcos para chegar na fluência

Alicerce

Marco 1 – Assistir aula inaugural

Marco 2 – Realizar todas as atividades – Postar resumo/fazer perguntas das aulas no repositório github = roger-melo-treinamentos/curso-de-js-roger-melo

Marco 3 – Como realizar as atividades: 0 – Faça os exercícios – 2 – Faça os exercícios corrigidos junto com o professor até que vc entenda os procedimentos – Implemente novamente, sozinho, do jeito do professor – 3 – Se tiver dúvidas, procure tirá-las usando suas anotações, e tente fazer sozinho novamente do zero.

Fazer uma revisão raio-x antes de cada implementar cada aplicação – Tem link explicando

Validação

Marco 4 – Análise das aplicações pelo professor

Marco 5 – Montar um portifólio com as 3 primeiras aplicações, no github e website. Pelo jeito o Roger não sabe da existência do github pages por citou que as pessoas podem não saber mexer com o github e baixar os seus projetos

Marco 6 – Se candidatar as vagas – Quando o medo é maior que o desejo vc paralisa, senão vc age. A cura do medo é ação

Evolução

Marco 7 – Implementar os feedbacks dos recrutadores

Marco 8 – Ser contratado

Comemoração

Marco 9 – Postar sua comemoração no repositório do treinamento

Plantão da fluência. atendimento individualizado de questões não técnicas

# Aula 002

01:05

Objetivos do treinamento

# Aula 003

03:01

Definindo JS

# Aula 004

07:10

Configurando o ambiente – VSCode – A extensão Live Server

<html lang=”pt-br”>

Instalar o f.flux inibe a luz azul do Monitor para descansar a visão

VSCode ctrl + shift + p -> Atalho para todas as funções

VSCode ctrl + b -> Fecha/Abre a aba de arquivos

Emmet ! <tab> -> Estrutura html

# Aula 005

03:36

alert(‘Mensagem’)

Mensagem popup no browser com botão OK

<script src=”App.js”></script>

# Aula 006

02:28

Chrome f12 -> Abrir aba Devtools

Configurar o Chrome DevTools com temas diferentes

DevTools -> ‘3 pontinhos’ -> Setings -> Aparencia -> Theme

console.log(‘mensagem’) -> exibir mensagens em terminais ou no console do devtools no chrome

Javascript é executado de forma sequencial de cima para baixo

# Aula 007

09:11

Constante, variáveis e comentários

let nomeVar

const nomeVar2

var nomeVar3 -> var não é mais usado

console.log(nomeVar, ‘Texto’, nomeVar2, nomeVar3)

let e const vieram no JS5

regras de nomeação de variáveis - camelCase

// Comentário de 1 linha

/\*

Cometários de várias linhas

\*/

VSCode ctrl + ; Coloca/retira comentários de linha

# Aula 008

04:44

Primeiros exercícios

No github – Em um olhinho escrito watch vc pode clicar nele e selecionar uwatch para ser avisado de alteração naquele repositório

Instalação do git

Instalação do terminal Hyper feito em html css js

hyper.is

git –version

git version 1.9.1

VSCode shift + ctrl + e -> Só abre a aba arquivos, não fecha

# Aula 009

08:04

Correção dos exercícios da aula 001

# Aula 010

03:21

Overview sobre tipos de dados

Number, String, Boolean, Undefined, Null, Object, Symbol, BigInt

Null -> quando nós queremos deixar explicitamente uma variável sem valor

Undefined -> O JS declara quando uma variável não possui valor e nem foi declarada com null

# Aula 011

12:52

Strings

“um texto normal”

‘um texto normal’

`texto ${variável} texto texto ${variável}`

‘texto’ + “texto2” + ‘texto’

textotexto2texto

Concatenação de strings

“texto”.length -> exibe a quantidade de caracteres

let novo = ‘texto’

console.log(novo[1])

e

JS – Zero based

console.log(“texto”.toUpperCase())

TEXTO

console.log(“TEXTO”.toLowerCase())

texto

Strings possui métodos que agem em cima do objeto original e outros em cópias destas strings

O Professor declarou que as Strings são imutáveis porém depois citou que existe métodos que agem no original e outros que agem na cópia

console.log(“xxx@xxx.com”.indexOf(‘@’));

3

# Aula 012

08:06

Métodos comuns de strings

‘texto’.lastIndexOfA(‘t’)

3

Retorna o índice da última ocorrência do parâmetro na string

const email = 'laracroft@gmail.com'

console.log(email.slice(0, 9))

laracroft

Retorna a string entre o primeiro parâmetro até o segundo parâmetro

Pode ser feito desta forma tumém

const email = 'laracroft@gmail.com'

console.log(email.slice(0, email.indexOf('@')))

laracroft

Não usando o Segundo parâmetro é retornado a string entre o valor do parâmetro usado até o fim da string

const email = 'laracroft@gmail.com'

console.log(email.slice(9))

@gmail.com

string.replace(‘string’,’string’)

const email = 'laracroft@gmail.com'

console.log(email,'\n',email.replace('lara', 'sarada'))

laracroft@gmail.com

saradacroft@gmail.com

Troca a primeira ocorrência da string do primeiro parâmetro pela do segundo parâmetro

# Aula 013

11:34

Numbers

Operadores aritméticos

/ -> Divisão

% -> Modulo

Área de um círculo

const radius = 10

const pi = 3.14

const area= pi \* radius \*\* 2

‘pi vezes (\*) radius Elevado (\*\*) ao quadrado (2)’

Ordem de precendência

1 -> Parêntese

2 -> expoentes (exponenciação) ou raízes

3 -> Multiplicação e divisão

4 -> Adição e subtração

Ou seja

radius \*\* 2 -> Foi resolvido primeiro

pi \* ‘resultado da expressão acima’

Incremento e decremento

let temp = 10;

console.log(temp++);

console.log(temp)

console.log('--------------')

temp = 10;

console.log(++temp);

console.log(temp)

console.log('--------------')

Resultado:

10

11

--------------

11

11

--------------

-------------------------------------------

Addition/Subtraction/mult/div assignment

destination assignment

let temp = 10

temp += 10

cosole.log(temp)

20

= temp = temp + 10

------------------------------------

NaN

# Aula 014

09:12

Template strings ou templates literals

`${variavel} texto texto `${variavel} texto `${variavel}`

\’ -> Escapando aspa simples

|” -> Escapando aspando enters duplas

\n ->Escapando enter

\t -> Escapando tabulação

Crase ou Back strict?

Templates html

const html =

`

<h2>${posTitle}$</h2>

<p>Autor: ${posAutor}$</p>

<span>Este post contém: ${postComents} comentarios</span>

`

Ai é só usar no html com .textHtml = html

# Aula 015

1,29

Exercícios 2

# Aula 016

27:56

Correção dos exercícios 2

EMMET script[src=”./app.js”] <tab>

# Aula 017

10:10

Arrays (objetos array)

let heroes =['batman', 'catwoman', 'iron man']

console.log(heroes)

[ 'batman', 'catwoman', 'iron man' ]

heroes[3] = 'Spiderman'

console.log(heroes)

[ 'batman', 'catwoman', 'iron man', 'Spiderman' ]

heroes[0] = 'BobSponja'

console.log(heroes)

[ 'BobSponja', 'catwoman', 'iron man', 'Spiderman' ]

É possível armazenar dados de tipos diferentes em arrays, porém o array foi criado para armazenar dados que possuem uma relação entre si, e o professor acredita que isso significa dados do mesmo tipo

let arrayRandom = [100, 'catwoman', 'iron man',13 ]

Propriedades e métodos

heroes.length -> exibe a quantidade de itens

console.log(heroes.join())

BobSponja,catwoman,iron man,Spiderman

console.log(heroes.join('---'))

BobSponja---catwoman---iron man---Spiderman

console.log(heroes.indexOf('catwoman'))

1

console.log(heroes.concat(['hombre raspadinha', 'melão das trevas']))

[

'BobSponja',

'catwoman',

'iron man',

'Spiderman',

'hombre raspadinha',

'melão das trevas'

]

heroes.concat(...) – return um novo array

console.log(heroes.push('he-man', 'Descarga fatal')) -> Não foi necessário colocar os itens entre colchetes

6

Porque retornou 6? porque o push altera o array original e return a nova quantidade de elementos. Se colocar novamente:

console.log(heroes)

[

'BobSponja',

'catwoman',

'iron man',

'Spiderman',

'he-man',

'Descarga fatal'

]

Observe que apareceu os novos itens incluídos com o push mas não os itens incluídos com o concat porque o concat return um novo array e para visualizá-los teríamos que guardar o return do concat em uma variável

console.log(heroes.pop())

Descarga fatal

heroes.pop() exclui último elemento do array original e return o elemento excluído

console.log(heroes)

[ 'BobSponja', 'catwoman', 'iron man', 'Spiderman', 'he-man' ]

O 'Descarga fatal' não existe mais

# Aula 018

04:23

null -> atribuído intencionalmente em variáveis que não possuir conteúdo – O null convertido em string return 0 (zero) por isso null + 10 é igual a 10

undefined -> quando o js não localiza o conteúdo da variável que deveria ter conteúdo. Undefined + 10 é igual a NaN

# Aula 019

01:25

Exercícios 3

# Aula 020

19:50

Correção exercícios 3

Sempre verificar o retorno de cada expressão e funções/métodos antes de usar, nem que seja apenas para vc saber e assim trabalhar de forma correta. Porém, em muitos casos esta verificação deve ocorrer também em sua implementação

# Aula 021

11:40

Booleans true/false

const email = 'brucewayne@xxx.com'

const include = email.includes('@') -> Retorna true/false se o parâmetro existe dentro da string email

console.log(include)

true

O método ‘string’.includes(‘caracter procurado’) também pode ser usado em arrays, no caso o ‘caracter procurado’ deve corresponder a um valor do array, deixando claro que o valor deve ser exato ao que se procura e não apenas parte do nome do valor. Exemplo:

const names = ['carlos', 'elaine', 'davi']

let arrayInclude = names.includes('elaine')

console.log(arrayInclude)

true

arrayInclude = names.includes('elai')

console.log(arrayInclude)

false

Comparações entres strings com > < = etc

Leva em conta:

inicias finais do alfabeto sempre serão Maiores > que as letras iniciais do alfabeto

Letras minúsculas são sempre maiores que letras maiúsculas mesmo sendo ‘iguais’

console.log('bela' > 'roger')

false

console.log('bela' > 'Roger')

true

console.log('bela' > 'Bela')

true

# Aula 022

05:36

Booleans true/false

== -> Compara apenas valores

=== -> Compara valores e tipos

const age = 31

console.log(age == 31)

true

console.log(age == '31')

true

console.log(age === ‘31’)

false

Isso acontece porque o JS faz conversão implícita de dados. Para evitar problemas sempre use os operadores ‘exclusivos’ ( comparação estrita ) que comparam valores e tipos e não faz conversões

===

!==

# Aula 023

08:02

Conversão de tipos

let score = '100'

console.log(score + 1 + '\n')

console.log(typeof score + '\n')

score = Number(score)

console.log(score + 1 + '\n')

console.log(typeof score + '\n')

1001

string

101

number

Segundo o professor, o typeof não é a melhor escolha para se definir o tipo de algum elemento

Conversão de um string para número que contiver letras ou símbolos da erro retornando NaN

String(97)

Boolean(10)

Retorna true

Valores falsy – Que retornam false

false

0 – Zero

“” – ‘’ - ``

null

undefined

NaN

Valores truthy

Todos os que não são falsy

# Aula 024

20:11

Segunda etapa – Controle de fluxo – Só citado

Resolução dos exercícios 4

Boolean( null + 1 )

null é avaliado como zero, então ficaria

Boolean( 0 + 1 )

true -> Porque 1 é true

# Aula 025

01:32

O que é controle de fluxo – Explicação

loops

Estruturas condicionais

# Aula 026

12:05

for loop

for(let i = 0; i < 10; i++){

console.log(i);

}

console.log('Cabô o for');

# Aula 027

05:47

whilw loop

let i = 0

while(i < 10){

console.log(i);

i++

}

console.log('Cabô o while');

# Aula 028

19:01

Correção exercícios aula 01

Fluxo - loops

# Aula 029

05:11

if – Primeira parte, sem o else

# Aula 030

05:45

if – Segunda parte – else e else if

# Aula 031

07:12

Operadores lógicos ‘ou’ || e ‘e’ &&

# Aula 032

25:01

Correção aula passada

const hour = 10

const isMorning = hour >=6 && hour <= 11

// Entre 6 e 11

const isAfternoon = hour >=12 && hour <= 17

// Entre 12 e 18

if(isMorning){

console.log('Bom dia')

}else if(isAfternoon){

console.log('Boa tarde')

}else{

console.log('Boa noite')

}

// O else, 'boa noite' será executado sempre

// que não estiver entre isMorning e isAfternoon

// isso significa, 23 -97 4400 210 etc





O Código fica maior, mas fica mais legível



Estes três itens fazem a mesma coisa

ings `)

Último exercício



odd = ímpares

even = pares

# Aula 033

03:22

Operador lógico not ( ! )

Inverte o boolean

let isUserLoggedIn = false

if(!isUserLoggedIn){

// !isUserLoogedIn não tem seu valor reatribuído

// o resultado da expressão com o ! que é usado

console.log('você precisa fazer login para continuar')

}

console.log(true);

console.log(false);

// true

// false

console.log(!true);

console.log(!false);

// false

// true

# Aula 034

03:52

Break e continue

Relacionado a loops

// Exibindo pontuações de um jogo

// Pontuação mínima, 0, não deve ser exibida

// Pontuação máxima atingida, não deve ser

// exibido scores posteriores a pontuação

// máxima

const scores = [50,25,0,30,100,20,10]

let scoresShow = []

console.log('Pontuações: ', scores.toString())

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

// Pontuação mínima

if( scores[i] === 0){

console.log('Você atingiu a pontuação máxima');

continue

};

// Pontuação máxima

if(scores[i] === 100){

break

}

// Coninuando loop - Pontuação

scoresShow.push(scores[i])

}

console.log('Pontuações exibidas: ', scoresShow.toString())

Pontuações: 50,25,0,30,100,20,10

Você atingiu a pontuação máxima

Pontuações exibidas: 50,25,30

# Aula 035

A condicional switch e exercícios

05:54

O switch sem o break exibe todas as opções após encontrar o resultado da condição

switch(condição){

case ‘A’:

console.log(“A”)

break

case ‘B’:

console.log(“A”)

Não tem break no default porque ele só vai ser ativado se nenhuma case for válido

break

default:

console.log(“???”)

}

# Aula 036

Correção dos exercícios da Aula 03 da etapa 02

29:16

Usar uma const dentro de um loop é possível porque o valor da variável do tipo const não é alterado e sim a variável const é recriada a cada iteração

Oque é mais performático recriar a const ou alterar uma let definida fora do loop?

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

const word = vetor[i]

}

# Aula 037

Funções – Apresentação

02:41

# Aula 038

Funções

06:46

Function declaration

function sayHi(){

...

}

Function expression.

const showFood = function(){

…

}

Há apenas uma diferença entre estes métodos de criação de função, o hosting

Usando o método declarativo podemos criar a função em qualquer lugar e chama-la normalmente, mesmo antes de sua criação

O mesmo não acontece com as functions expressions, elas, OBRIGATÓRIAMENTE, só podem ser executadas abaixo do código da criação da função

Para evitar surpresas, sempre declarem as funções no topo do código

# Aula 039

Funções: Argumentos, parâmetros e default parameters

06:28

Na criação da função usamos PARÂMETROS

Na execução da função usamos ARGUMENTOS

A ordem dos parâmetros deve corresponder a ordem dos argumentos passados, já o uso dos argumentos no código da função não tem relação com a ordem em que foram recebidos

function oi(name, lastName){

console.log(lastName)

console.log(name)

}

oi(name, lastName)

Se não fosse colocado os argumentos eles iriam receber ‘undefined’

Por isso, podemos definir parâmetros com valores default

function oi(name = ‘Elaine’, lastName = ‘Luiz’){

console.log(lastName)

console.log(name)

}

oi()

Elaine

Luiz

oi(‘Carlos’, ‘Arruda’)

Carlos

Arruda

# Aula 040

Funções: Retornando valores

06:39

# Aula 041

Correção de exercícios

17:53

const funcoes = [

function(){ return 'Ola'},

function(){

return 'Meu'

},

()=>'Mundo'

]

let frase = ''

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

frase += `${funcoes[i]()} `

}

console.log(frase)

Ola Meu Mundo

# Aula 042

Arrow functions

04:48

const drobo1 = (number = 0) => {

return 2 \* number

}

const drobo2 = number => {

return 2 \* number

}

// Na versão acima não é possível

// atribuição de parâmetro default

let number =0

const drobo3 = () => {

return 2 \* number

}

const drobo4 = number => 2 \* number

//const drobo5 = number => if(2===2)console.log('Dois')

// Desta forma não pode retornar uma expressão

// Achei uma exceção

const drobo5 = () => 3 > 4 ? 'sim':'não'

console.log(drobo5())

# Aula 043

Funções e Métodos

03:39

Métodos estão associados a objetos ou classes e antes de serem chamados é colocado um ponto ( . ), ou seja, fora de objetos e classes ( Talvez também em outras estruturas ) usamos funções, dentro, usamos métodos

const name = ‘roger’

console.log( name.toUpperCase() )

ROGER

# Aula 044

forEach e callbacks

07:14

callbacks são funções que recebem outra(s) funções como argumentos

const myFunc = callback => {

const value = 77

callback( value )

}

myFunc( number => console.log(number) )

77

//-------------------------------------

const socialNetworks = [ 'youtube', 'twitter', 'instagram', 'facebook']

socialNetworks.forEach( () => {

console.log('Ola')

})

Ola

Ola

Ola

Ola

Fazendo uma pequena alteração

socialNetworks.forEach( socialNetwork => {

console.log(socialNetwork)

})

youtube

twitter

instagram

facebook

Fazendo mais algumas alterações

socialNetworks.forEach( (socialNetwork, index) => {

console.log(`${index} - ${socialNetwork}`)

})

0 - youtube

1 - twitter

2 - instagram

3 - facebook

Fazendo mais algumas alterações

socialNetworks.forEach( (socialNetwork, index, array) => {

console.log(`${index} - ${socialNetwork} - ${array}`)

})

0 - youtube - youtube,twitter,instagram,facebook

1 - twitter - youtube,twitter,instagram,facebook

2 - instagram - youtube,twitter,instagram,facebook

3 - facebook - youtube,twitter,instagram,facebook

E ainda podemos fazer

const logArrayInfo = (socialNetwork, index, array) => {

console.log(`${index} - ${socialNetwork} - ${array}`)

}

socialNetworks.forEach( logArrayInfo )

Com o mesmo resultado da última alteração

# Aula 045

Utilizando callbacks

04:31

index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Utilizando callbacks</title>

</head>

<body>

<h1>Utilizando callbacks</h1>

<ul data-js="ul"></ul>

<script src="./app.js"></script>

</body>

</html>

app.js

const ul = document.querySelector(`[data-js="ul"]`)

const socialNetworks = [ 'youtube', 'twitter', 'instagram', 'facebook']

let HTMLTemplate = ''

socialNetworks.forEach(socialNetwork => {

HTMLTemplate +=`<li style="color: deepink;"> ${socialNetwork} </li>`

})

ul.innerHTML = HTMLTemplate

# Aula 046

Correção de exercícios – Aula 02 – Etapa 03

30:40

O ‘return’ de uma função pode substituir o ‘break’ da estrutura ‘switch’

const temp = []

const likes = (names = []) => {

switch( names.length){

case 0:

return 'Nínguém curtiu'

case 1:

return '1 curtiu'

case 2:

return '2 curtiu'

default:

return 'bugou'

}

}

console.log(likes())

temp.push('a')

console.log(temp)

console.log(likes(temp))

temp.push('b')

console.log(temp)

console.log(likes(temp))

temp.push('c')

console.log(temp)

console.log(likes(temp))

console.log(temp)

Nínguém curtiu

[ 'a' ]

1 curtiu

[ 'a', 'b' ]

2 curtiu

[ 'a', 'b', 'c' ]

bugou

[ 'a', 'b', 'c' ]

# Aula 047

O que são objetos

02:12

# Aula 048

Criação de um objeto literal ( objetos individuais )

08:01

propriedade: valor

chave: valor

let user = {

name: 'João',

idade: 48,

preferencias: ['cha','café', 'cevada'],

exibir: ()=>console.log(user.name)

}

console.log(user)

console.log(user.name)

console.log(user.preferencias[2])

user.sexo = 'm'

console.log(user)

user.exibir();

console.log(user[‘idade’])

Resultado:

{

name: 'João',

idade: 48,

preferencias: [ 'cha', 'café', 'cevada' ],

exibir: [Function: exibir]

}

João

cevada

{

name: 'João',

idade: 48,

preferencias: [ 'cha', 'café', 'cevada' ],

exibir: [Function: exibir],

sexo: 'm'

}

João

48

# Aula 049

Adicionar métodos em um objeto

02:40

let show = {

exibir: function(){

console.log(user.name)

}

}

let show = {

exibir: () => {

console.log(user.name)

}

}

let show = {

exibir: () => console.log(user.name)

}

show.cabelo = ()=>console.log('preto')

# Aula 050

Aula 5 – Correção de exercícios

array.includes(valor)

Não inclui algo no array, ele VERIFICA a existência de determinado VALOR no array retornando TRUE ou FALSE

True = existe

False = não existe

const letra = ['a','b','c','d']

console.log(letra.includes('a'))

console.log(letra.includes('f'))

console.log(['a','b','c','d'].includes('a'))

console.log(['a','b','c','d'].includes('f'))

//--------------------------------------------------

const getTypeDefinition = type => {

const obj = {

null: 'Seta, explicitamente, uma variável sem valor',

undefined: 'Representa um valor não-setado (não atribuido)',

object: 'Arrays, Datas, Objetos literais, Funções, etc'

}

return obj

}

console.log(getTypeDefinition())

Resultado:

{

null: 'Seta, explicitamente, uma variável sem valor',

undefined: 'Representa um valor não-setado (não atribuido)',

object: 'Arrays, Datas, Objetos literais, Funções, etc'

}

//--------------------------------------------------

const getTypeDefinition2 = type => {

const obj = {

null: 'Seta, explicitamente, uma variável sem valor',

undefined: 'Representa um valor não-setado (não atribuido)',

object: 'Arrays, Datas, Objetos literais, Funções, etc'

}

return obj[type]

}

console.log(getTypeDefinition2('null'))

console.log(getTypeDefinition2('undefined'))

console.log(getTypeDefinition2('object'))

Resultado:

Seta, explicitamente, uma variável sem valor

Representa um valor não-setado (não atribuido)

Arrays, Datas, Objetos literais, Funções, etc

//--------------------------------------------------

const getTypeDefinition2 = type => {

return {

null: 'Seta, explicitamente, uma variável sem valor',

undefined: 'Representa um valor não-setado (não atribuido)',

object: 'Arrays, Datas, Objetos literais, Funções, etc'

}[ type ]

}

console.log(getTypeDefinition2('null'))

console.log(getTypeDefinition2('undefined'))

console.log(getTypeDefinition2('object'))

Resultado:

Seta, explicitamente, uma variável sem valor

Representa um valor não-setado (não atribuido)

Arrays, Datas, Objetos literais, Funções, etc

# Aula 051

Aula6 variáveis e escopo de bloco

09:35

Uma variável declarada fora de um bloco é global – isso é ruim

Dentro de blocos de códigos o escopo é local

declarar/criar uma variável é diferente atribuir um valor a dita cuja

O professor demonstrou sua predileção a const invés de let

var chamais

# Aula 052

A palavra chave this

08:14

this, um objeto de contexto

this pode ser usada apenas dentro de objetos e classes

Dentro de um objeto/classes o this é uma referência aos dados deste mesmo objeto/classe

por isso usar o this em locais diferentes, por exemplo, em um navegador vc obtém resultados

específicos dos locais onde o this foi chamado

IMPORTANTE dentro do objeto/classe quando vc declarar uma função use apenas as functions declarations...

...

chamar: function ( ){

...

}

...

...porque em uma arrow function o this não será uma referência para o próprio objeto, e sim uma referência a raiz do app, no navegador, o objeto Windows

Existe um meio para utilizar uma sintaxe mais curta ainda mantendo a função do this

...

Basta retirar os dois pontos ( : ) e a palavra function

chamar ( ){

...

}

...

# Aula 053

Aula8 correção de exercícios

13:10

# Aula 054

Aula9 Objetos em arrays

04:04

Armazenar objetos em um array é comum

# Aula 055

Aula10 O objeto Math

07:15

Math é um objeto embutido no Javascript com PROPRIEDADES e MÉTODOS

Math.PI – Retorna o pi

Math.E – retorna a base dos logaritmos naturais

Math.round(7.7) – retorna o inteiro 8 – do .5 em diante

Math.round(7.4) – retorna o inteiro 7

Math. floor (7.7) – retorna o inteiro 7 – sempre arredonda para baixo

Math.floor(7.4) – retorna o inteiro 7

Math.ceil(7.4) – retorna o inteiro 8 – sempre arredonda para cima

Math.trunc(7.4) – retorna o inteiro 7 – apenas retira a parte decimal

Math.random() – retorna um número aleatório entre 0 e 1

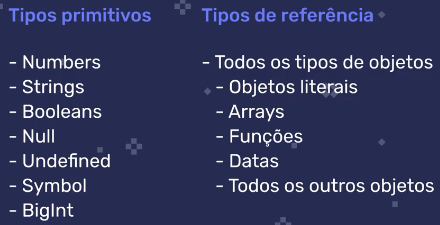
Math.round(Math.random()) – Retorna 1 ou 0 aleatóriamente

Math.round(Math.random() \* 100) – Retorna um número de 0 a 100

# Aula 056

Aula11 tipos de referência

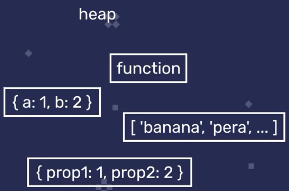
11:29

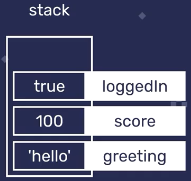


A diferença entre eles está na forma como eles são armazenados na memória

stack ou pilha - É um espaço limitado mas mais rápido. É aqui que ficam armazenados os tipos primitivos

heap ou amontoado É um espaço mais amplo porém mais lento



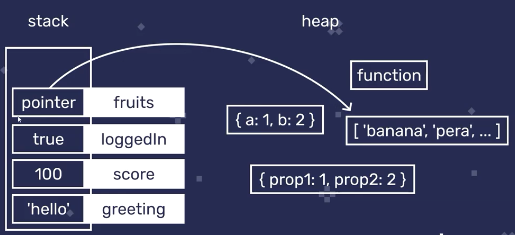


Nós acessamos os valores na stack através de uma variável

loggedIn acessa true

score acessa 100

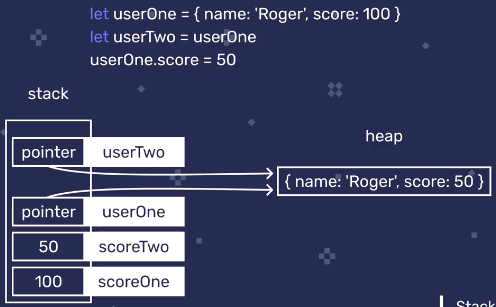
greeying acessa ‘hello’



A memória heap é acessada através de um ponteiro e este ponteiro através de uma variável. Este ponteiro fica gravado na stack



Criamos a variável com o identificador scoreOne e atribuímos o valor 50 para ela. Em seguida criamos a variável scoreTwo e atribuímos a ela a variável scoreOne. Neste ponto ambas as variáveis estão com o valor 50. Em algum momento alteramos o valor da scoreOne para 100, ISTO NÃO ALTERA O VALOR da scoreTwo pois cada variável armazena seu valor em locais diferentes dentro da stack



# Aula 057

Aula1 Correção de exercícios

12:07

Macete com uma template string

let message = 'ola mundo'

Observe o espaço da margem esquerda até o hífen...

for(let i = 0;i < 5; i++){

message += `

- ${message}`

}

console.log(message)

Resultado:

...o espaço da margem esquerda até o hífen é colocado no resultado da template string. Parecendocom o uso da tag <pre> em html

ola mundo

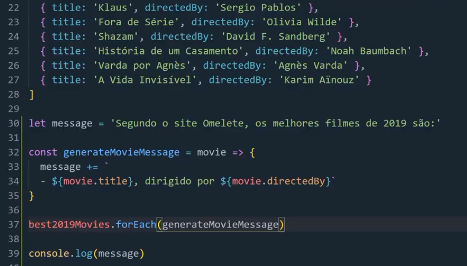
- ola mundo

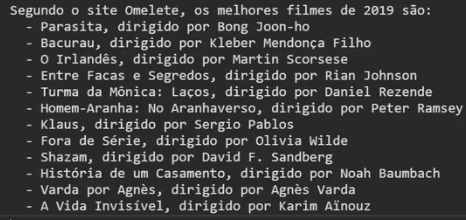
- ola mundo

- ola mundo

- ola mundo

...





# Aula 058

Aula2 Mais alguns escopos

13:40

1 – Escopo de funções

2 – Escopo de blocos

3 – Escopo léxico

var let const

No ES6 em diante a var tem um comportamento indesejado

if(true){

let dragon = 'Balerion'

Aqui funciona

Aqui da erro porque dragon esta declarado dentro do bloco if

console.log('Dentro do if: ',dragon);

}

console.log('Fora do if: ',dragon);

Se mudarmos a palavra chave let para var....

if(true){

var dragon = 'Balerion'

console.log('Dentro do if: ',dragon);

}

A visibilidade das variáveis declaradas com var são ultra globais. apenas dentro de funções elas não eram visíveis globalmente

console.log('Fora do if: ',dragon);

Resultado:

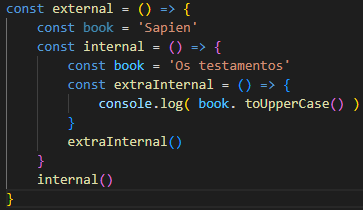
Dentro do if: Balerion

Fora do if: Balerion

--------------------------

Escopo lexico

Quando vc usa funções dentro de funções, funções aninhadas, as variáveis contidas na função pai podem ser visualizadas nas funções filhas. O contrario não é válido, as variáveis declaradas nas funções filhas não podem ser vistas na função pai. Quando a função pai e a filha declararem variáveis com o mesmo nome o resultado depende de quem vai ser executada, se for a pai pega a variável declarada na Pai e se for executada a filha pega da filha. Neste ultimo caso é preciso ter cuidado para dentro da pai não ter a execução da filha... a busca por qual variável usada percorre o caminho do aninhamento

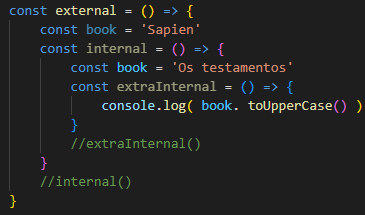
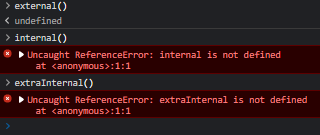


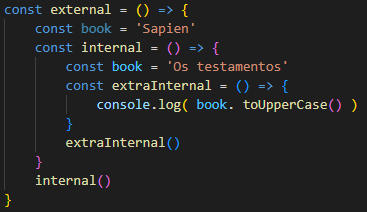
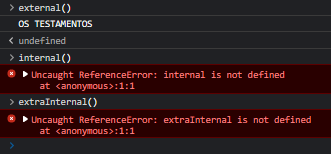
A busca pela variável percorre este caminho?

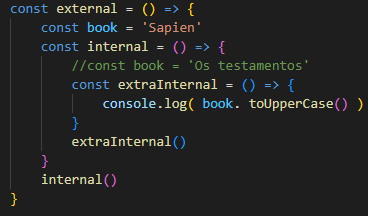
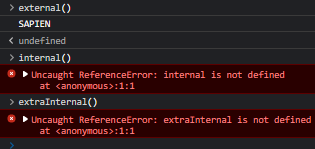
Como extraInternal é que tem a chamada para o console, ela tem que ser executada, e tem que ser dentro da internal, que é onde ela esta. E a internal também tem que ser executada porque a extraInternal está dentro dela. A internal deve ser executada no escopo da external.

Por fim vc executa a external() no console do navegador para exibir algum o resultado

No node dá erro para tudo que é lado







# Aula 059

Aula3 Documentação

02:57

MDN

Buscador do MDN ineficiente

No google slice mdn

Prefira a versão em english

# Aula 060

Aula4 Interagindo com o Browser

01:38

Adicionar conteúdo

Modificar css

Reagindo a eventos

Criar interações menos dscretas

# Aula 061

DOM Document Object Model

06:31

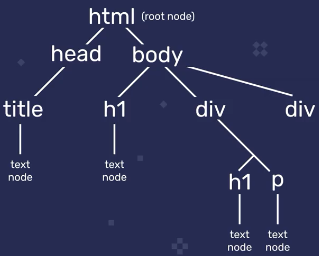
Criado pelo browser

document modelagem do documento html

Digitando no console. browser

document.location

document.URL



# Aula 062

Aula6 QuerySelectors

10:16

querySelector e querySelectorAll são as formas mais eficientes de se trabalhar com o DOM

**querySelector** seleciona a primeira tag encontra e ignora as outras

- const paragraph = document.querySelector(/‘p’)

<p>Hello World<p>

- const paragraph = document.querySelector(‘.error’)

<p class=”error”>Essa é uma mensagem de erro</p>

- const paragraph = document.querySelector(‘#error’)

<p id=”error”>Essa é uma mensagem de erro</p>

- const divError = document.querySelector(‘div.error’)

<div class=”error”>Essa é uma mensagem de erro</div>

- const titulo = document.querySelector(‘body > h1’)

<h1>Essa é uma mensagem de erro</h1>

**querySelectorAll** seleciona todas as tags encontradas e as armazena em um NoList

- const paragraph = document.querySelectorAll(‘p’)

NodeList(3) [ p, p, p.error ]

NoList parece um array mas não tem todas as propriedades e métodos de um. Da para usar um for each

# Aula 063

Aula7 Outras querys

06:13

- const title = document.getElementById(‘titulo’)

<h1 id=”titulo”>Title</h1>

- const erros = documentgeElementsByClassName(‘error’)

HTMLCollection [ p.error, p.error ]

HTMLCollection é uma lista genérica de elementos, parecido com o NodeList, não igual. Por exemplo, não é possível dar um for each

- const paragraphs = document.getElementsByTagName(‘p’)

HTMLCollection [ p.error, p.error, p, p ]

----------------

O professor prefere usar os querySelector por causa da flexibilidade, pode-se usar as tags css, e porque retornam um NodeList que possuem mais propriedades e métodos que os HTMLColletion, principalmente por causa do uso do for each

# Aula 064

Aula8 Correção dos exercícios

17:48

const getCatInfo = ( ) => {

const nome = ‘Marcos

let age = 3

const color = ‘Cinza

}

Para retornar as informações do gatos é necessário retornar, return, od dados

const getCatInfo = ( ) => {

const name = ‘Marcos

let age = 3

const color = ‘Cinza

return { name, age, color } // Short sintax

// Uma observação sobre a shorth sintax, se houver argumento de mesmo nome

// ele seria chamado e não o valor da ‘propriedade’

}

// Desestruturando o objeto retornado da getCatInfo

const { name, age, color } = getCatInfo()

console.log( name, age, color )

----------------------------

array.reverse() // Inverte os elementos do array, age sobre o original

----------------------------

console.log( arrays.shift( ) ) // Deleta o primeiro item do array, age sobre o original

----------------------------



-----------------------------

# Aula 065

# Aula 066

# Aula 067

# Aula 068

# Aula 069

# Aula 070