Anotações da primeira aula ao vivo Otterwise

O discord é onde toda a comunicação é feita.

O portal é onde teremos os códigos e o review das aulas ao vivo.

Turma 05 formação web developer

Aprovação e desempenho, tem que entregar tudo que for solicitado pela equipe e precisamos ter 70% de aprovação nos projetos e no projeto final.

Monitorias fixas basta chegar

O algoritmo é uma maneira que a programação resolve problemas, é uma abstração para resolver um problema.

Ou seja, o algoritmo nada mais é que a solução.

A lista de compras do mercado é um algoritmo.

O JS é uma linguagem de programação interpretada onde ela irá ler, linha por linha.

CTL + J para abrir o terminal

Usaremos o node para rodar o nosso programa.

O cd é o comando que utilizamos para navegar por pastas dentro do terminal

O ls mostrará tudo que tem dentro da pasta

O node é a ferramenta que interpreta o JS para a gente.

Toda a linguagem de programação precisa de uma ferramenta para interpretar a nossa linguagem e o node fará isso

O node main irá rodar o nosso primeiro programa

Declaramos uma variável no JS da seguinte forma:

Let  
const  
var

Variáveis em linguagem de programação elas armazenam uma informação e guardam na memória.

No JS // é o comentário de uma linha

As variáveis nos ajudam a ter uma manutenção de código mais saudável

Ao chamarmos gaveta no console, estaremos na verdade instanciando os valores.

Se tentarmos alterar o valor de uma constante iremos ter um erro retornado pelo console, chamado de typeError, pq não podemos alterar o valor de uma constante

Boa pratica começar as variáveis com letra minúscula e sem espaço ou utilize o camelCase, não coloque tb caracteres especiais.

Os tipos primitivos, são os tipos que nós podemos ter dentro do JS.

String

Number

boolean

Undefined

Null

\n indica quebra de linha.

Números fracionados utilizamos o “.” E não a “,”.

Booleans: True e false

Nunca atribui o valor undefined para uma variável

Ex:

Let gaveta = undefined

Jamais faça isso.

Para colocarmos o espaço entre variáveis utilizamos as “ ”

O + irá fazer a concatenação

Console.log(“Olá, eu sou o joão e tenho “ + 27 + “anos”)

Ele vai transformar o 27 em string pois ao ler a linha ele havia identificado incialmente que se tratava de uma string.

Console.log(27 + “10”)

Aqui o retorno será 2710 ele irá concatenar tudo e transformar em uma string e não irá realizar a soma como se poderia esperar, pois não podemos somar um número com uma string.

Expressões aritméticas

Soma

Multiplicação

Subtração

Divisão

Resto

potência

Sobrecarga de operador é quando o mesmo operador faz duas coisas e depende muito do que você está tentando fazer, se você tiver dois números o + irá somar, caso contrário ele irá transformar em uma string.

A precedência é exatamente a mesma da matemática básica que aqui é feita com a utilização dos ()

Condicionais

If

Else

Else if

Default do comportamento padrao

Quando usamos === iremos comparar o valor e também o tipo, se usarmos somente == iremos comparar somente os valores.

Switch Case

Case

Break

default

Expressões lógicas

And = &&

Or = ||

Negative = !

**Anotações aula 02**

Function soma (argumentos)

Argumento normalmente é algo interno a função

Soma (parâmetros)

Qualquer valor pode ser passado como parâmetro de função (que seja válido no JS)

A função é uma estrutura para automatizarmos nossos processos.

O JS sempre joga as suas variáveis até o topo do escopo que ele está (VAR). O mesmo acontece com a function (hosting)

Quando temos o let ele não é jogado para cima.

É sempre bom usarmos uma const quando não formos alterar mais o valor dessa atribuição.

Exercicios:

1. Faça uma função que recebe um valor inteiro e verifica se o valor é par. A função deve retornar um valor booleano se for par.

2. Faça uma função que recebe a idade de um nadador e retorna a categoria desse nadador de acordo com a tabela abaixo:

Idade - Categoria

5 a 7 anos - Infantil A

8 a 10 anos - Infantil B

11 - 13 anos - Juvenil A

14 - 17 anos - Juvenil B

Maiores de 18 anos (inclusive) – Adulto

**Estrutura de repetição**

A mais comum de todas é o for

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

}

**Listas**

Listas são nada mais que um gaveteiro, onde cada uma dessas gavetas você pode colocar algo la, a lista é uma coleção de elementos que chamamos de array.

Dentro do array poderemos colocar qualquer tipo de elemento.

Nós podemos colocar quantos elementos quisermos dentro da lista. (O máximo que a memória aguentar e esse tamanho é gigantestico)

let num = 0

let numArray = [10, 20, 30, 40]

console.log(numArray)

A primeira posição de um array é necessariamente 0

let num = 0

let numArray = [10, 20, 30, 40]

console.log(numArray)

console.log(numArray[1] + numArray[0])

let fruits = ['banana', 'maçã', 'laranja']

console.log(fruits[2])

function hello() {

  console.log('Hello world')

}

let fruisAndNumbers = ['banana', 2, 'maçã', 4, null, undefined, true, hello]

console.log(fruisAndNumbers[7]())

const fruits = ['laranja', 'banana', 'maçã']

for (let index = 0; index < fruits.length; index++) {

  const element = fruits[index]

  console.log(element)

}

**Object**

Um objeto no JS é totalmente diferente do objeto da orientação objeto de outras linguagens.

Assim definimos um objeto da seguinte forma: Ele é um elemento

Imagine que vamos definir uma pessoa

const name = 'João'

const age = 27

const person = {

  name: 'joao',

  age: 27,

}

console.log(person.name)

console.log(person.age)

Podemos ter um objeto dentro de um objeto, pq um objeto é um valor válido dentro do JS

Veja:

const name = 'João'

const age = 27

const person = {

  name: 'joao',

  age: 27,

  address: {

    street: 'Nome da minha rua',

    Number: 19023,

  },

}

console.log(person.name)

console.log(person.age)

console.log(person.address)

Texto

Descrição gerada automaticamente

const name = 'João'

const age = 27

const updateDate = 'data de atualização'

const person = {

  name: 'joao',

  age: 27,

  address: {

    street: 'Nome da minha rua',

    Number: 19023,

  },

  'data-de-criacao': '14/12/2021', // Quando temos uma propriedade que esta em "" devemos colocar o colchete

  [updateDate]: '14/12/2021',

}

console.log(person.name)

console.log(person.age)

console.log(person.address)

console.log(person['data-de-criacao'])

console.log(person[updateDate])

Texto

Descrição gerada automaticamente

const name = 'João'

const age = 27

const updateDate = 'data de atualização'

const person = {

  name: 'joao',

  age: 27,

  address: {

    street: 'Nome da minha rua',

    Number: 19023,

  },

  'data-de-criacao': '14/12/2021', // Quando temos uma propriedade que esta em "" devemos colocar o colchete

  [updateDate]: '14/12/2021',

  competencias: ['javascript', 'css', 'html'],

}

console.log(person.name)

console.log(person.age)

console.log(person.address)

console.log(person['data-de-criacao'])

console.log(person[updateDate])

console.log(person.competencias[0])

Texto

Descrição gerada automaticamente

Quando temos uma variável e elas tem o mesmo contexto, é ideal assim utilizarmos um objeto

O JS nos deixa mudar o valor de uma constante se ela for um array ou um objeto

**Booleanos primitivos**

Todos os tipos primitivos do JS avaliam para true or false de alguma forma

const test = true

if (test) {

  console.log('deu bom')

}

const string = "" // aqui ele será avaliado para falso

if(string) {

    console.log("deu bom")

} // retornará nada, pois a string vazia é setada para falso.

Não só são as strings que são avaliados para falso

O null como poderia se esperar era o null e o undefined

O 0 evidenemte é falso do aberto é fechado do binário

Classes especiais. As classes são utilitários de um mesmo contexto.

O not a number ele não avalia para falso, para isso podemos usar uma função para avaliar (agoa avalia)

isNaN() >> isso é uma classe

console.log(**new** *Date*())

como date é uma classe, nós precisamos do new

Quando criamos uma data, de forma automática ele cria o time zone 0.

Todos os tipos primitivos tem as suas classes.

O Object também um tipo primitivo

Array.isArray([]) é a única forma de sabermos se uma variável é um array.

**String**

const firstName = 'João'

const lastName = 'bretanha'

console.log(firstName + ' ' + lastName)

console.log(`${firstName} ${lastName}`)

Template String

Split da string

const frase = 'Olá, meu nome é joão'

console.log(frase.split(' ')) // ele irá separar uma string com espaço e retornará um array de string

1. Crie um algoritmo que tem como entrada um array de números e imprime no console a soma dos elementos sendo cada um deles multiplicado pelo seu índice.

Exemplo entrada: [5, 9, 10, 6]

Exemplo Saída: 47

2. Crie um algoritmo que tem como entrada um objeto e imprime no console os nomes dos projetos ativos do usuário.

user = {

name: "Juca",

projects: [

{ name: "Projeto 1", start: "01/02/2021" , active: true},

{name: "Projeto 2", start: "03/03/2021", active: false},

{name: "Projeto 3", start: "10/08/2021", active: true},

{name: "Projeto 4", start: "20/08/2021", active: false},

{name: "Projeto 5", start: "18/10/2021", active: true}

]

}

3. Levando em consideração o array [6, 21, 9, 2, 50, 98, 1] crie uma função que mostra o maior numero da lista

4. Crie uma função que conte quantas palavras existem na frase que for passada como parâmetro (dica: utilizem o método split de string)

5. Através do array de usuários abaixo imprima no console todas a skills que cada usuario tem:

const users = [

{

name: "Joao",

skills: ["Javascript", "ReactJS", "Redux"]

},

{

name: "Pedro",

skills: ["VueJS", "Ruby on Rails", "Elixir"]

}

]

6. Crie uma função chamada rockPaperScissorsWinner e recebe dois valores como parametro. Esses dois valores podem ser:

0: tesoura

1: papel

2: pedra

Construa um algoritmo que recebe esses valores randomicamente e printa na tela qual usuario ganhou, se o usuario 1 ou usario 2

**Aula 04: Teste de Mesa**

O teste mesa é uma ferramenta para acompanharmos o que está acontecendo com o nosso projeto

Retomar 3 conceitos

**Aula 07**

O ideal de uma função é que ela funcione por si só, que ela não dependa de coisas externas a aela. Exemplo

function sonma(*a*, *b*) {

  return *a* + *b*

}

let c = 3

console.log(soma(1, 2)) // temos uma depeência externa a função

Acima temos uma função impura, a função pura independente dos valores, ela precisa passar o mesmo resultado. Não é o caso na imagem acima.

O certo era o C ir para dentro da função via parâmetro.

//Função impura

function soma(*a*, *b*) {

  return *a* + *b* + c

}

let c = 3

console.log(soma(1, 2)) // temos uma depeência externa a função

// Função pura

function soma2(*a*, *b*, *c*) {

  return *a* + *b*

}

console.log(soma2(1, 2, 3))

Forma alternativa de declarar uma função

const soma3 = function (*a*, *b*) {

    return *a* + *b*

}

Uma função é um valor válido em JS, o que significa que podemos passar uma função como parâmetro no JS, e uma convenção é chamar ela de call-back

function show(*value*) {

  console.log(*value*)

}

function soma4(*a*, *b*, *callback*) {

  const total = *a* + *b*

  callback(total)

  return total

}

soma4(2, 2, show)

Uma função pode retornar outra função

function sumTwoNumbers(*numbeOne*) {

  return function (*numberTwo*) {

    return *numbeOne* + *numberTwo*

  }

}

console.log(sumTwoNumbers(5)(4))

Essa função de retorno só pode ser anônima.

const operations = {

  sum: function (*a*, *b*) {

    return *a* + *b*

  },

}

console.log(operations.sum(2, 2))

Um objeto também pode retornar uma função

**Funções em arrays**

const operationsArray = [soma]

console.log(operationsArray[0](4, 1))

**Arrow Functions**

const multiply = (*a*, *b*) => *a* \* *b*

console.log(multiply(2, 5))

Exercicios:

Crie uma função que recebe duas notas como argumento e retorna a média entre elas. Utilize arrow function e atribua a uma constante.

**If ternário**

if (value >= 10) {

  console.log('Valor é igual ou maior que 10')

} else {

  console.log('valor é menor que 10')

}

// if ternário

console.log(

  value >= 10 ? 'valor é igual ou maior que 10' : 'valor é menor que 10'

)

console.log(

  value > 10

    ? 'Valor é maior que 10'

    : value === 10

    ? 'valor é 10'

    : 'valor é menor que 10'

)

**Destructiring assignemnt**

É fazer uma atribuição de forma resumida

Ele veio usado para não termos que ficarmos chamando o nome do nosso objeto

const person = {

  name: 'joao',

  age: 27,

  address: {

    street: 'minha rua',

    number: 102,

  },

}

const {

  name,

  age,

*address*: { street, number },

  address,

} = person

console.log(name)

console.log(age)

console.log(street)

console.log(number)

console.log(address)

const person = {

  name: 'joao',

  age: 27,

  address: {

    street: 'minha rua',

    number: 102,

  },

}

const {

  name,

  age,

*address*: { street, number },

  address,

} = person

console.log(name)

console.log(age)

console.log(street)

console.log(number)

console.log(address)

const names = ['Joao', 'Pedro']

const numbers = [1, 2]

const [firstName, secondName] = names

const [, numberTwo] = numbers

console.log(firstName)

console.log(secondName)

console.log(numberTwo)

const users = ['joao', 'Rodrigo', 'christian', 'sidemar']

const getFirstElement = ([*FirstElement*]) => {

  return *FirstElement*

}

console.log('getFirsElement', getFirstElement(users))

1. Crie uma função que recebe uma string (com quatro possibilidades: “soma”, “subtrai”, “divide”, “multiplica”) e dois números. A função deve retornar a operação realizada informada pela string nos dois números.

Exemplo Entrada:

myFunction(‘soma’)(3)(5)

Exemplo Saída:

8

2. Crie uma função que recebe um objeto usuário e imprime se ele é maior de idade, utilize os conhecimentos da aula para melhorar o código.

Exemplo Entrada:

{ name: ‘Juca’, idade: 28 }

Exemplo Saída:

Maior de idade

Aula 10

Uma string é um interable igual a um array

Uma string pode ser manipulada utilizando alguns métodos, onde nós podemos iterar sobre ela, pegar uma parte etc.

>> Métodos de string

Slice: Pega uma parte da string

Nós podemos pelo slice começar de um determinado ponto da string e terminar

O Fim da contagem é excluído, ou seja, se o fim definido for 12, ele vai até 11. Aqui o final é opcional. Você pode pegar de um determinado ponto e ir até o final caso você não determine um fim.

Substring: ele também espera um número inicial e um número final só que aqui somos obrigados a passar um parâmetro, no slice não. O slice pode ser utilizado em arrays, o substring não.

Substr: é uma função depreciada pelo JS, mas podemos utilizar, aqui o conceito é a partir de qual posição que você quer, e o seu tamanho. O ideal é não utilizarmos essa função

>> Métodos para modificar conteúdo de uma string

Replace: Mudar ou trocar um valor um por outro de uma string

replaceAll ele pode pegar toda as palavras que você passou como parâmetro p avaliar e trocar, no replace ele troca apenas a primeira.

O I na regex /Otterwise/i case insensitve

O g troca todas as ocorrências da minha palavra

toUpperCase: irá tornar todas as palavras maiúsculas

toLowerCase: faz justamente o inverso do UpperCase

trim: remove os espaços desnecessários no início e no final de uma senteça

padStart e padEnd:

padStart, nós podemos definir que o tamanho precisa ser 4 e caso ele não tenha esse tamanho com 4, você pode definir o que será preenchido caso não esteja completo. Aqui ele preenche depois o padEnd preenche depois.

CSV: command separeted value

Um método alternativo o Slice quando temos uma string muito grande separada por “,” como um CSV, você utiliza o split, você pode dividir ele por “,” e ele irá retornar para você um array com as informações que você precisa. Esse método se chama split

**>> Métodos de Array**

// Adicionar novos valores em Array

//Concat: irá inserir novos valores dentro do array. Quando utilizamos o concat ele irá retornar um array novo, atenção a isso. O array original não será atualizado. Aqui podemos concatenar tanto uma string quanto um array

O Push irá inserir também elementos no array, aqui o array original será alterado.

Também podemos transformar um array em uma string de elementos, através do método joined

//Pop: Remove o último elemento de um array

O pop também modifica o array principal, cuidado.

//Shift: Remove o primeiro elemento do array

Ele também modifica o array principal, removendo o primeiro

//unshift: coloca algo no início do array

Ele também modifica um array principal

// slice: cada posição do array será contada, cuidado ele é exclusivo, então se o final for 1, ele vai parar a exibição no 0, se for 2 ele para no 1... Ele não modifica o array principal.

// Splice usamos para adicionar ou remover elementos do array

>> Removendo:

Tools.splice(1,1) >>> A partir de 1 remover o 1.

Aqui ele irá modificar o array principal

Ele é uma maneira de remover o array do meio.

>> Adicionando:

Tools.splice(0, 0, “Chave de boca”, “martelete”)

Aqui ele irá adicionar chave de boca e matelete na posição 0, deixando os elementos p trás sem removê-los

Se colocarmos o 1 depois do primeiro 0, ele irá excluir um elemento.

// Reverse: Podemos inverter a posição do nosso array

O array é modificado, ele inverte a ordem inteira do array

Todos os métofos que modificam um array tem um caso de uso, mas cuidado ao utilizar, o ideal é não modificar e sim criar um array novo.

>> Métodos de iteração

forEach((element) => {

}) ele espera que se passe uma função como parâmetro.

forEach((user, index, array) => {

console.log(user)

console.log(index)

console.log(array)

})

Todos os demais exemplos usam o index e o array também.

// Map : Aqui você pode mapear o valor de um elemento a outro.

Users.map((user) => {

Return {

Name:user,}

})

A diferença do map para o foreach você não precisa retornar nada, o map já pede que você retorne o novo valor desse elemento.

Ele mapeia um tipo de elemento para outro.

Vamos usar o map quando é necessário mudar os valores de um array

// Filter é quando queremos filtrar um elemento.

O filter sempre espera que você retorne true or false

Eles retornam um novo array sem tocar no array original (map, filter, reduce)

// Find é um método que iremos encontrar apenas a nossa informação.

Ele irá retornar um true or false, igual o filter

O find retorna o elemento, o filter irá retornar um array de elementos.

O find ele quer retornar apenas uma ocorrência, então ele acha a primeira e para, se você quiser uma lista ai sim a solução correta é utilizar o filter.

// everey >>> ele irá verificar se todos satisfazem o parâmetro que você informar

Ele é muito semelhante ao find, o Every retorna sempre um boolean.

// some ele irá procurar se existe algum elemento dentro daquele array baseado no parâmetro passado.

Aqui o retorno também é um boolean

// sort aqui iremos ordenar um array. Para sortear um número nós temos que passar uma função de comparação.

// Reduce: Queremos reduzir um array que recebemos em alguma informação

No reduce ao contrário dos outros métodos ele recebe um acumulador, não um valor que vai passando a cada iteração, no reduce ele é o segundo argumento reduce(acummulator, num)

Const sum = numbers.reduce((acc,num) => {

Return acc + num;

}, 0\*)

\*primeiro valor a ser passado para comparação.

>>> GIT

O primeiro conceito de git é um repositório, logo ela está dentro do controle de versão

Uma branch é basicamente uma linha do tempo, precisamos saber qual a linha do tempo ela está seguindo.

Os checkpoints são os nossos commits, e isso é dizer ao git que aquilo é um checkpoint

A ação de juntar duas branches em uma só é chamado de merge request ou pull request, mas o nome correto é um merge.

Comandos:

Para startar um repositório git, precisamos que seja da pasta raíz.

Git status mostra o status de todos os nossos arquivos

Untracked files ele está dizendo que ele não reconhece essas pastas ainda

O git add . ele vai adicionar todas as pastas da pasta que você ta para o seu commit

Git commit -m “Nosso primeiro commit“ ... max 72 carac.

A flag m significa, message

Git checkout nome\_da\_branch

O git fetch ele olha as alterações que tem no remoto e você não tem. Serve p o repositório local comparar no repositório remoto e saber se tem diferença entre eles.

Git pull origin nome\_da\_branch

O JSON pode ser um dados e também ser salvo em um formato de arquivo.

{

“name”: pode ser objeto, string, numero, array etc

}

Não separamos json por virgula

{

“id”: = 0,

“name”: “biblioteca”,

“books”: [“Comunicação não violenta”]

}

Reutilizando código sem com funções, que é isolar ele em arquivos

E usaremos o import e o export

export const hello() => “hello world”

em outro arquivo

import { hello } from “./functions.js”

para usarmos import e export precisamos utilizar os módulos no JS e para usar os módulos nós precisamos utilizar gerenciadores de pacote, que utilizamos o npm do node

O yarn é um outro gerenciador de pacotes muito popular, e possui pequenas diferenças em relação ao npm

**Criando um projeto utilizando o npm**

O packet.json quer dizer que nós temos um workspace

Então precisamos agora criar nossos projetos iniciando com o npm init

Ao colcarmos ao só colocarmos o export na frente, isso se chama um export nomeado, contudo nós podemos exportar de outra forma além do nomeado, que seria o export default a diferença é que não podemos mais importar com import { hello } agora iremos por fora das chaves e podemos por o nome que desejarmos

Import hello from

Os módulos nós temos o comum js

O pack-lock.json ele tem a função de garantir a integridade do pacote que nós estamos utilizando.

Lembrar sempre de por o node\_modules no gitIgnore.

Install [date-fns@2.27](mailto:date-fns@2.27) o @ é para podermos escolher qual a versão nós iremos utilizar, caso contrário iremos instalar a última versão estável disponível.

Uma API é uma interface de comunicações

Uma requisição o JS lida com requisições utilizando as promises.

Uma promise nada mais é que uma promessa e é como o JS lida com uma função assíncrona, um código assíncrono é algo que não acontece na hora, pois ele precisa ir e voltar no servidor. O js já tem implementado essas promises que nos da a possibilidade de implementar essas funções assíncronas

Status:

Pendente

Sucesso

Rejeitado ou erro

A promise precisa estar dentro de uma função, pq ele precisa retornar uma função

Async Await

Para utilizarmos o await a função mais acima dela tem q ter o label async (função pai)

Métodos HTTP

O get iremos buscar por algum recurso

O método HTTP se resume hoje ao get e put (atualiza o post) ou post (adiciona elemento)

O PUT troca todas as informações que você mandou no back e o path só parte dele.

Rest e graphicL

O padrão REST é orientada a recurso

Vamos utilizar promises para nos comunicarmos com a API

<https://jsonplaceholder.typicode.com/> >>> Fake Api

para usar ela, precisamos utilizar um cliente HTTP

Temos uma biblioteca p fazer essas requisições chamada de axios

Npm add axios

Um endpoint nada mais é o domínio e o recurso é o que será apresentado depois da / da endereço

O response retorna 4 status

**ESTUDO DE CASO**

Iremos analisar um problema, e iremos resolver esse problema juntos.

ESTUDO DE CASO 01 – 20/01/2022

Quando digitamos o cep em um site e ele completa sozinho, é feita uma requisição através de uma API para buscar esses dados no cadastro.

Proposta do estudo de caso

Npm add console-input (para termos a leitura do usuário digitar)

Para realizarmos uma requisição, precisamos instalar o axios e para realizar requisições em API, precisamos criar uma requisição assíncrona e para chamarmos utilizamos o método get()

A POC (prova de conceito) nada mais é que a gente testar coisas que nós pensamos em nossa aplicação por parte.

Desenvolva o seu sistema por partes.