**JAVASCRIPT**

**OUTPUT:**

* Manipulando um item html pelo seu id através do JS;

Document.getElementById(“id”).innerHTML = “novo texto”;

* Janela de alerta da página;

Window.alert(“mensagem”);

* Exibir a mensagem no console;

console.log(‘o que quiser’);

**VARIÁVEIS:**

* Use o “let” ou “var” pra criar variáveis.
* Var pode ser usada em qualquer escopo global
* Let só pode ser usada no escopo local, não se vê ele fora do código, por exemplo, no navegador.
* Não pode começar com números
* Frases tem que estar entre parênteses
* Número não precisa
* Pra aparecer as informações no console, digite ‘console.log(variável)’
* ‘const’ é uma constante, uma variável que não pode ser modificada, é útil na utilização de arrays e objetos.
* Concatenação; adição, juntar, somar.

**IF E ELSE:**

* > maior
* < menor
* == igual
* === exatamente igual
* != diferente
* !== totalmente diferente
* >= maior ou igual
* <= menor ou igual
* && E, and
* || Ou, or

**CONDICIONAL TERNÁRIA:**

* Let isMember = true;

Let shipping = isMember ? 2 : 10;

Desmembrando:

isMember **{deu certo?}** ? **{então é 2}** 2 **{deu errado?}** : **{então é 10}** 10;

* ? se sim
* : caso contrário

**SWITCH:**

* let profession= “policial”

switch (profession) {

case ‘fiscal’ :

console.log(sua camisa será verde);

break;

case ‘policial’ :

console.log(‘Sua camisa será azul);

break;

default:

console.log(‘sua camisa será preta)

break;

Desmembrando:

switch (profession) **{a variável que será analisada}** {

case **{caso seja}** ‘fiscal’ :

console.log(sua camisa será verde);

break **{acabou esta case aqui}** ;

default **{normal, caso não satisfaça nenhuma das anteriores}** :

console.log(sua camisa será preta)

break;

}

* switch; troque, trocar
* case; caso, caso seja, if, true
* break: quebrar, separar, terminar
* default; padrão, else

**FUNÇÕES:**

**Arrow Function:** Diferença pra função normal é deixar o código mais limpo, ex;

Function;

Function somar(x, y) {

Return x + y;

}

Arrow Function;

Const somar = (x, y) => x + y;

**Variável de escopo global**: criada fora da função, pode ser usada em qualquer function

**Variável de escopo local**: criada dentro da função, é usada somente naquela situação;

**Função dentro de uma função:**

Function addPtc(a, b) {

Const square = (x) +> x \* x; **[aqui não precisa ser arrow function mas fica melhor pra não se perder]**

Let ptcA = square(a);

Let ptcB = square(B);

Return ptcA + ptcB;

}

**ARRAY´S**

**Lista de algo**

Como criar um array?

* Let colors = [‘blue’, ‘red’, ‘green’];
* Console.log(colors[1])

Vai exibir a cor **red**.

Da pra pegar as informações de um array e usar em outro? **Sim**

Da pra criar um array dentro de outro array? **Sim**

Como eu faço pra adicionar um item no array depois de tê-lo feito já?

* Nomedoarray.push(‘oquequeradd’);

Pop = remove o último item do array

Shift = remove o último item do array

Array.length = para visualizar quantos itens tem o array

Array[1] = visualizar o item que está no 1

**OBJECT´S**

**Grupo de informações**

Diferença dele pro array, nele eu posso dar características, ex;

Let personagem = {

Nome: ‘Arthur’,

Idade: 20,

Pais: ‘Brasil’

}

Como visualizar somente uma característica?

* personagem.nome

Da pra criar um objeto dentro de um objeto? **Sim, mesmo esquema do array.**

Da pra criar um array dentro de um objeto ou vice-versa? **Sim**

Mudar algum item:

Personagem.nome = ‘pedro’

Adicionar mais idade:

Personagem.idade += 10

Exemplo de um Objeto => Array => Objeto (lista de carros e cores deles:

Objeto = Azul

Array = Verde

Let personagem = {

Nome: ‘Arthur’,

Idade: 20,

Carros: [

{ modelo: ‘Fiat’, cor: ‘preto’ },

{ modelo: ‘Ferrari’, cor: ‘vermelha’ }

]

}

E pra eu acessar somente a cor do Fiat?

* Console.log(personagem.carro[0].cor);

Utilizando uma função dentro de um objeto:

Function: Verde

Let pessoa = {

Nome: ‘Arthur’,

Sobrenome: ‘miranda’,

Idade: 20

nomeCompleto: function () {

return `${this.nome} ${this.sobrenome};

}

Ou ao invés de function se quiser usar arrow function pode também, da seguinte maneira:

nomeCompleto: () => {

return ‘Arthur Miranda’;

}

**LOOP´S**

**Quando eu quero executar algo várias vezes.**

Como é demonstrado o loop? **For**

**Ex:**

for (let n = 1; n <= 10; n++) {

console.log(‘Arthur’ + n);

}

**Iria fazer isto:**

Arthur 1

Arthur 2

Arthur 3

Arthur 4

Arthur 5

Arthur 6

Arthur 7

Arthur 8

Arthur 9

Arthur 10

Laranja: por onde vai começar a minha contagem

Azul: Até onde vai a minha contagem

Roxo: Vai avançar de quanto em quanto (no caso ali de 1 em 1)

Vermelho: pra mostrar o número depois do arthur pra ficar mais intuitivo, mas não é necessário

**LOOP´S COM WHILE**

*Enquanto a condição for satisfeita o código vai continuar rodando*

* let c = 0;

while(c < 10) {

console.log(‘”Número: “+c’);

c++;

}

Vermelho: Condição;

Verde: c = c + 1... serve pro código não ficar rodando pra sempre pois não sairia do 0

**LOOP´S EM ARRAY**

Let cores = [

{ nome: ‘preto’, quantidade: 10 }

{ nome: ‘vermelho’, quantidade: 12 }

];

**Maneiras de usar o loop nesse array:**

for (let = 0; n < cores.lenght; n++) {

console.log(cores[n].quantidade[0]);

}

**Outra maneira:**

For (let i in cores) {

Console.log(cores[i].quantidade[0]);

}

**Outra maneira:**

For (let cor of cores) {

Console.log(cor.quantidade[0]);

}

Vermelho: usamos o cores.length pra ele dar loop na quantidade de vezes em que se temos objetos, por exemplo, ali seriam duas vezes, preto e vermelho.

Se eu quiser visualizar só a quantidade do vermelho? Adiciona o verde no código

**EVENTOS:**

Eventos de clique;

1°: Coloque na sua tag ‘button’ no html, o ‘onclick=”função”’

2°: No JS crie uma função para quando o evento for disparado o que você programou aconteça.

Ou

1°: Usar o ‘this’ pra mudar o próprio botão

2°: onclick=”this.innerHTML=’Clicou’”

Eventos de passar o mouse;

1°: Mesmo esquema do onclick, muda-se para ‘onmouseover=”função”’ ou ‘onmouseout=”função”’

2°: No JS crie uma função para quando o evento for disparado o que você programou aconteça.

Existem diversos de eventos com o mouse, se você digitar ‘on’ no vscode ele já te dará diversas opções de eventos.

Eventos de teclado;

1°: Coloque na sua tag ‘button’ no html, o ‘onkey=”função”’, ai pode ser, onkeyup. onkeydown e onkeypress

2°: No JS crie uma função para quando o evento for disparado o que você programou aconteça.

KeyboardEvent ex;

Onkeyup=”digitou(event)”

* ‘event’ funciona tipo o ‘this’, que mostra quais teclas estão sendo apertadas naquele evento.

*Exemplo/Dica:*

Eu quero que o usuário digite algo, e quando ele apertar o **enter**, isto apareça no meu console...

* Function digitou(e) {

If (e.keyCode == 13) {

Let texto = document.getElementById(“campo”).value;

Console.log(‘texto’)

}

}

Azul: usamos para determinar que é um código de keyboard

Verde: código da tecla **enter**

Laranja: ID do campo que estava sendo digitado

Vermelho: Pra pegar o valor daquele campo específico

*Exemplo mais avançado:*

Tem que estar com o **enter** e o **ctrl** apertado...

* Function digitou(e) {

If (e.keyCode == 13 && e.ctrlKey == true) {

Let texto = document.getElementById(“campo”).value;

Console.log(‘texto’)

}

}

Roxo: porque não keyCode? Porque ctrl é assim e pronto o cara não explicou porque, mas está ai o exemplo.

**Usando botões no JS pra alterar o CSS:**

1°: Crie no seu CSS ‘class’ com o que você deseja alterar, neste exemplo, mudaremos a cor do H1.

2°: Crie no seu HTML buttons para referenciar cada uma das cores que desejará alterar, no nosso caso, vermelho e azul.

3°: Agora em cada button coloque a função que ele corresponderá.

4°: Agora crie as funções, azul e vermelho, e dentro delas usaremos os seguintes códigos;

* Limpar()
* Document.getElementById(“titulo”).classList.add(‘azul’ ou ‘vermelho’);

Azul: usaremos este para que quando o outro botão seja clicado, sejam excluídas as informações do botão anterior, mas, é uma function certo? Então que função será esta?

* Function limpar() {

Document.getElementById(‘titulo’).classList.remove(‘azul’ ou ‘vermelho’)

}

Laranja: Id do nosso item no HTML que está sendo trocado de cor.

Verde: é pra acessar a lista de classes do css no js.

Rosa: tag usada pra ADICIONAR a cor.

Cinza: tag usada pra REMOVER a cor.

*Outro exemplo;*

Quero Mostrar o número de telefone só depois de eu clicar no botão...

1°: Lá no CSS você pega o id da div na qual está situado o seu botão, e escreve que você quer ‘display:none’ pra o número ficar oculto.

2°: Crie um botão

3°: Agora no seu JS crie a seguinte função;

* Function mostrarTelefone() {

Document.getElementById(telefone”).style.display = ‘block’;

}

*E se eu quiser que o botão suma depois de eu aperta-lo?*

4°: Function mostrarTelefone(**elemento**) {

**Elemento.stlye.display = ‘none’;**

Document.getElementById(telefone”).style.display = ‘block’;

}

E lá no HTML dentro do ‘onclick = mostrarTelefone(**this**)’ adiciono o this...

**DOM = Document Object Model**

* document.getElementById(‘id’).**innerHTML** = “Arthur”;

*Esse comando irá pegar o elemento que está identificado com o id que selecionei e por conta do* ***innerHTML****, que quer dizer ‘****dentro do htm****l’ e irá adicionar o item que vem depois nas aspas.*

* document.getElementsByClassName(‘class’)[0].innerHTML = ‘item alterado’;

*Vai selecionar as classes que selecionei, se escreve no plural, porque se pode ter mais de uma class com o mesmo nome, ela sempre te retornará um array.*

* Document.getElementsByTagName(‘tag’)

*Vai selecionar todas as tags desejadas*

* Document.getElementsByName(‘nome’)

*Menos utilizado...*

* Document.querySelector(‘’)

*Substitui todos os acima, usando ‘#’ pra id´s e ‘.’ Pra classes*

* Document.querySelectorAll(‘’)

*Seleciona todos os itens se tiver mais de um, em forma de array*

Criando uma função para mudar a cor de uma div quando o botão for apertado;

1°Crie os botões que servirão como gatilho para as funções

2°Coloque neles a função onclick=’cor de cada um’

3°Crie no css uma seção para cada uma das cores

4°Crie a div que mudará de cor

5°Crie as funções abaixo

Function azul() {

Document.querySelector(‘div’).**classList**.**add**(‘azul’);

}

Function vermelho() {

Document.querySelector(‘div’).classList.add(‘vermelho’);

}

Vermelho: Acessa a lista de classes do css

Azul: adiciona a classe a seguir na div

*Trocar o ‘add’ por* ***‘contains’*** *vai verificar se há aquela classe naquele elemento.*

**TIPOS DE DADOS**

Var nome = “arthur” -> **String**

Var idade = “20” -> **Number**

Var save = true -> **Boolean**

Var cidade -> **Undefined**

*Exemplo: ‘typeof arthur -> string’*

**PARÂMETROS**

Funções do próprio jS;

* function alterar(titulo) {

document.getElementById(“titulo”).innerHTML = titulo;

}

Alterar(‘mudei o titulo’)

* function somar(x, y) {

let total = x + y;

document.getElementById(“titulo”).innerHTML = total;

}

Somar(‘2, 2’);

Ou...

* function somar(x, y) {

let total = x + y;

return total;

}

var resultado (‘10, 15’)

console.log(‘resultado’)

**MANIPULANDO ATRIBUTOS**

*Mudando a foto (gato e cachorro) da página através de um botão;*

1°: crie a tag ‘img’ pra alocar a imagem principal, neste caso do cachorro

2°: crie dois buttons, um pro cachorro e outro para o gato

3°: coloque no button a função ‘onclick=”trocarImagem(‘cachorro.jpg’)

e no outro botão substitua pelo gato.jpg.

4°crie a função abaixo:

function trocarImagem(filename) {

document.querySelector(.imagem’).setAttribute(‘src’, ‘images/’+filename);

}

Vermelho: tag para setar a troca de atributos

Azul: o atributo que eu desejo trocar

Verde: como vai ser mudado

*Agora quero que a página identifique qual animal está aparecendo.*

0°: dentro da tag img, adicione ‘data-animal=”cachorro”’

1°: adicione dentro do ‘onclick=”trocarImagem(‘cachorro.jpg’, ‘cachorro’)’

2°: crie outro button com o ‘onclick=”pegarAnimal()”’

3°: crie uma outra função como abaixo:

function pegarAnimal() {

let animal = document.querySelector(‘.imagem’).getAttribute(‘data-animal’);

alert(“O animal é: “+animal);

}

Laranja: adicionados onde já tinha coisa

Roxo: seleciona um atributo

**DIMENSÕES**

* document.querySelector(‘.texto’).offsetHeight ou Width

*offset vai mostrar qual a dimensão do item escolhido*

* document.querySelector(‘.texto’).clientHeight ou Width

*é descontado a barra de rolagem e borda, deixa só a largura e o padding*

* document.querySelector(‘.texto’).scrollHeight ou Width

*tamanho real do conteúdo*

**SCROLL**

* document.querySelector(‘.texto’).scrollTop ou Left

*vai mostrar em que px está localizado o scroll naquele momento*

* document.querySelector(‘.texto’).scrollTo(0, 0)

*via mandar o scroll pro px que eu mandar*

**MÉTODOS DE STRING**

*Quantos carácteres tem na variável ‘nome’?*

* let nome = ‘Arthur’

console.log(nome.lenght);

*Quero saber em que posição está o ‘carvalho’;*

* let nome = ‘Arthur de Carvalho’

console.log(nome.indexOf(‘Carvalho’))

**lhe retornará o número 10, caso ele não ache o que eu estou procurando, irá retornar -1.**

*Pegar um pedaço da String;*

* let nome = ‘arthur de carvalho’;

console.log(nome.slice(0, 10);

Vermelho: ‘cortar, partir, tirar um pedaço’

Azul: De onde até onde queres tirar o pedaço.

EXTRA: a tag ‘**substring**’ faz a mesma coisa que o ‘slice’, porém usando o slice possibilita você começar do final, por ex; nome.slice(-4);

**EXTRA**: a tag ‘substr’ faz parecido, mas quando você for dizer a posição, tem que fazer assim; nome.substr(8, 2); que significa que começou na posição 8 e vai contar duas casas pra frente. (O MELHOR MÉTODO)

*Substituindo textos de string;*

* let nome = ‘arthur miranda’

console.log(nome.replace(‘arthur’, ‘luís’)

lhe retornará ‘luís miranda’

Vermelho: mudar de lugar, trocar, substituir

Azul: Substituindo ‘x’ por ‘y’

*Sequência de funções:*

toUpperCase = vai pra maiúscula

toLowerCase = tudo minúsculo

concat = vai concatenar com alguma coisa (meio inútil)

trim = tira os espaços em branco da string, por ex, arthur miranda, ela vai tirar esse espaço entre o arthur e o miranda, serve legal pra usar em lugares que um usuário vai digitar algo.

charAt(x) = quando eu quero saber qual caractere está na posição x

split(‘ ‘) = onde você achar espaços (não precisa necessariamente ser espaço, pode colocar outra coisa dentro das aspas, e irá transformar em array.

**MÉTODOS DE NUMBERS**

* let n = 10.3453

let res = n.**ABAIXO**

*ABAIXO*:

toString = transforma o número em string

toFixed(X) = Reduz um número decimal pra quantidade de casas após a vírgula de X

parseInt(n) = transforma o n de string em número

parsFloat(n) = mesma coisa do parseInt mas ele preserva os números decimais

**MÉTODOS EM ARRAY´S**

* let lista = [‘Ovo’, ‘Farinha’, ‘Corante’, ‘Massa’];

let res = lista.**ABAIXO**

*ABAIXO:*

toString = transforma o array em string, botando vírgula entre os itens

join(x) = vai fazer a mesma coisa que o toString mas ao invés de vírgula ele vai separar os itens pelo que está no X

indexOf(‘x’) = vai me mostrar em que posição está o meu X dentro do array, se não achar retorna –1

*Os itens a seguir são colocados fora da lista porque alteram o próprio array e não o seu valor então ficaria assim;*

* let lista [‘ovo’, ‘farinha’, ‘corante’, ‘massa’];

lista.**ABAIXO**;

let res = lista;

*ABAIXO:*

pop() = remove o último item do array

shift() = remove o primeiro item do array

push(x) = adiciona o item X no meu array

[0] = ‘x’ = vai adicionar na posição 0 o item X, se já estiver ocupado ele troca de lugar com o que está atualmente

splice(x, y) = a partir de X eu quero remover a quantidade y

concat(lista2) = vai concatenar os array´s

sort = mostra o array em ordem alfabética

reverse = reverte na ordem que tá, se colocar o sort antes ele mostra em ordem contrária a alfabética

*Métodos um pouco mais avançados*

Função Map;

* let lista = [2, 3, 4, 5, 6]

let lista2 = []

lista2 = lista.map(function(item) {

return item \* 2;

})

let res = lista2

Vermelho: Irá passar item por item do array e executará a função em cada um deles

Azul: Em cada um dos itens irá executar uma função que retorna o número vezes 2

*Outra forma de fazer a mesma coisa usando loop´s em array*

* for(let i in lista) {

lista.push(lista[i] \* 2)

Função Filter;

* let lista = [2, 11, 4, 12, 6]

let lista2 = []

lista2 = lista.filter(function(item){

if(item > 10) {

return true;

}else {

return false;

})

Vermelho: irá filtrar os itens maiores que 10 e deixará o array somente com eles

Função Every;

* let lista = [2, 11, 4, 12, 6]

let lista2 = []

lista2 = lista.every(function(item){

if(item > 10) {

return true;

}else {

return false;

})

Vermelho: se todos os itens não satisfazerem a função ela retorna false

ADICIONAL: tem uma função muito parecida ‘some’ que dai se pelo menos 1 item satisfazer a função ele já retorna true

Função Find;

* let lista = [2, 11, 4, 12, 6]

let lista2 = []

lista2 = lista.find(function(item){

if(item == 11) {

return true;

}else {

return false;

})

Vermelho: Caso ache algum item igual a 11 retornará aquele item específico

ADICIONAL: tem uma função muito parecida ‘findIndex’ onde ao invés de ele retornar o item ele vai retornar a posição do item.

**DATE EM JAVASCRIPT**

* let d = new Date();

\*se o item fica vazio, ele pega o horário do seu pc ou da pessoa que está acessando\*

Contudo você pode preencher o item da seguinte forma:

(ano, mês, dia, hora, minuto, segundo)

Lembrando que o mês começa no 0 (janeiro)

Pra exibir, usamos o (d.toString());

Também pode configurar da seguinte forma:

(‘ano-mês-dia hora:minuto:segundo’)

Mínimo que você precisa preencher é ano e mês

*Manipulando as datas;*

* let d = new Date();

let novoValor = d.getYear();

Vai nos retornar o ano atual, a mesma coisa se eu usar, ‘getMonth’ ou ‘getDay’ ou ‘getDate’, Hours, Minutes, Seconds

Sobre o ‘getDay’, pegará o dia da semana, não do mês, semana começa no Domingo, que é representado pelo 0, ou seja, vai de 0 a 6

*Trocando o mês o ano ou a data;*

d.setFullYear(2040) = muda o ano pra 2040

d.setMonth(11) = muda o mês pra dezembro

d.setDate(14) = muda o dia pra dia 14

d.setDate(d.getDate() + 10) = aumenta o dia atual em 10 dias

**FUNÇÕES RELACIONADAS A MATEMÁTICA (Math)**

* let Valor = Math.etc(xx)

Math.round(x.xx) = arredonda o item

Math.floor(x.xx) = arredonda o item pra baixo

Math.ceil(x.xx) = arredonda o item pra cima

Math.abs(-xx.xxxx) = me mostra o absoluto deste item, ou seja, o positivo

Math.min(x, y, z) = me mostra qual dos itens é o menor

Math.max(x, y, z) = me mostra qual dos itens é o maior

Math.random() = me da um número aleatório entre 0 e 1

Math.floor(Math.random() \* 10)

* pegará um número de aleatório de 0 a 10 e garantirá que não passe de 10

**INTERVALOS (TIMERS)**

* let timer = setInterval(showTime, 1000)

Vermelho: Função de setar intervalos

Azul: Nossa função de relógio lá do ‘naula011’

Verde: tempo em que queremos que a função seja executada, mostrada em milissegundos, ou seja vai executar a função de 1 em 1 segundo

*E se eu quiser parar este timer?*

* clearInterval(timer);

Toda essa função pode ser vista no arquivo ‘naula011’ onde eu criei as funções, o botão pra começar o timer e pra finalizar, lá tem algumas informações extras

* setTimout

Basicamente via esperar tantos milissegundos e executará uma função, a atividade esta feita no arquivo ‘naula012’

*E se eu quiser impedir essa função?*

* clearTimeout(timer)

Timeout x Interval;

Timeout: Executa um única vez quando é mandado

Interval: Fica rodando até que seja mandado parar

**TEMPLATE STRING´S**

- O símbolo da template string é o ``

-É usado no lugar das aspas comuns pra poder alojar o ‘${x}’

Por Ex:

* let anos = 12

let nome = roberto

let frase = `Meu nome é ${nome} e eu tenho ${anos}`

Podemos até fazer contas matemáticas na frase;

* let anos = 20

let nome = arthur

let frase = `Meu nome é ${arthur} e ano que vem farei ${anos+1} anos`

**DESCONTRUÇÃO DE OBJETOS**

* let pessoa = {

nome: ‘Arthur’,

sobrenome: ‘Miranda’,

idade: 90,

nomeCompleto: function() {

return `${this.nome} ${this.sobrenome}`;

}

};

*Começando o processo de descontrução:*

let { nome, sobrenome, idade } = pessoa; 🡪

🡪Aqui então eu basicamente fiz uma variável que continha todas as informações que eu queria tirar do objeto, ao invés de ter que tirar uma por uma

*E se eu quiser que o nome da variável ‘idade’ não seja ‘idade’?*

let {nome, sobrenome, idade:pessoaIdade } = pessoa;

🡪Colocando o dois pontos e o novo nome eu altero o nome da variável pra que eu quiser

*Um pouco mais avançado:*

* let pessoa = {

nome: ‘Arthur’,

sobrenome: ‘Miranda’,

idade: 90,

social: {

facebook: ‘ArthurM’,

instagram: ‘ArthurMM’

}

nomeCompleto: function() {

return `${this.nome} ${this.sobrenome}`;

}

};

Se eu quiser desmembrar as informações do ’social’?

- let {facebook, instagram} = pessoa.social

Ou talvez eu queira o nome a idade e o instagram?

- let {nome, idade, social: {instagram} } = pessoa;

Azul: Outra Forma de fazer a mesma coisa

- function pegarNomeCompleto({nome, sobrenome}) {

return `${nome} ${sobrenome}`

}

Caso a pessoa não tenha sobrenome por exemplo:

- function pegarNomeCompleto({nome, sobrenome = Luyo}) {

return `${nome} ${sobrenome}`

}

Por padrão o sobrenome será luyo.

**DESCONTRUINDO ARRAY´S**

* let info = [‘arthur miranda’, ‘arthur’, ‘miranda’, ‘@arthur’];

let [ nomeCompleto, nome, sobrenome, instagram ] = info;

Assim ele irá pegar as informações, a desconstrução dos arrays é feita em ordem da sua criação original, seu eu quero por exemplo o primeiro e o último item, eu faço assim;

* let [nomeCompleto, , , Instagram] = info;

Deixando os espaços que eu não quero em branco.

Posso definir as variáveis de um array da seguinte forma também;

* let [nomeCompleto, nome, sobrenome, instagram] = [‘arthur miranda’, ‘arthur’, ‘miranda’, ‘@arthur’];

Outra forma, criando uma função e saindo dela as variáveis já desconstruídas;

* function criar() {

return [1, 2, 3]

}

Let [a, b, c] = criar();

Mesma coisa para objetos ^^

**OPERADOR DE SPREAD**

*Complementando um array;*

* let numeros = [1, 2, 3, 4]

let outros = [...numeros, 5, 6, 7]

*Complementando um objeto;*

* let info = {

nome: ‘Arthur’,

idade: 11,

sobrenome: ‘Mirandinha’

};

let novaInfo = {

...info,

cidade: ‘lages’

estado: ‘sc’

};

*Complementando uma função;*

* function adicionarInfo(info) {

let novasInfo = {

...info,

status: ‘available’

token: ’123454’

};

return novasInfo;

}

adicionarInfo({nome: ‘Arthur’, sobrenome: ‘Miranda’});

**OPERADOR REST**

* function adicionar(...numeros) {

console.log(numeros);

}

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

Se não tivesse o vermelho, não puxaria todos os números, somente o primeiro.

**FUNÇÕES INCLUDES E REPEAT**

*Em Array:*

* let lista = [‘ovo’, ‘carne’, ‘macarrão’]

console.log(lista.includes(‘ovo’) );

Se tiver o item no array retorna true, se não tiver, false

Não precisa ser a palavra, pode ser só o caractere

* console.log(‘x’.repeat(10) );

Vermelho: a função

Azul: o número de vezes que ela executará aquele elemento

**OBJETO: KEY, VALUES E ENTRIES**

* let pessoa = {

nome: ‘Arthur’,

sobrenome: ‘Miranda’

}

Keys:

* console.log( object.keys(pessoa) );

vai retornar: [nome, sobrenome]

Values:

* console.log( object.values(pessoa) );

vai retornar: [‘Arthur’, ‘Miranda’]

Entries:

* console.log( object.entries(pessoa) );

vai retornar: [ [ 'nome', 'Arthur' ], [ 'sobrenome', 'Miranda' ] ]

**STRING: PADSTART & PADEND**

*pedEnd:*

* let telefone = ‘5’;

console.log( telefone.padEnd(9, ‘\*’);

Vermelho: quantos carácteres tem que ter

Azul: O que colocar no lugar dos carácteres faltantes

*padStart* é igual, só que ele vai colocar os ‘asteriscos’ no começo ao invés do final

**REQUISIÇÕES**

* O que é: puxar uma informação externa

**JSON (JAVASCRIPT OBJECT NOTATION)**

* Objeto usado para fazer a comunicação entre requisições
* JSON.parse: transforma uma string em um objeto usável, ou seja, em um JSON
* JSON.stringfy: passa um JSON para uma string

Código Síncrono vs Código Assíncrono

* Síncrono:

Executa linha por linha

* Assíncrono:

Executa linha por linha, porém sem esperar retorno, só vai executando

Toda Requisição interna, é um código assíncrono

Callback´s

* É uma função JS que você cria e manda para outra execução
* O que é na prática?

function alertar () {

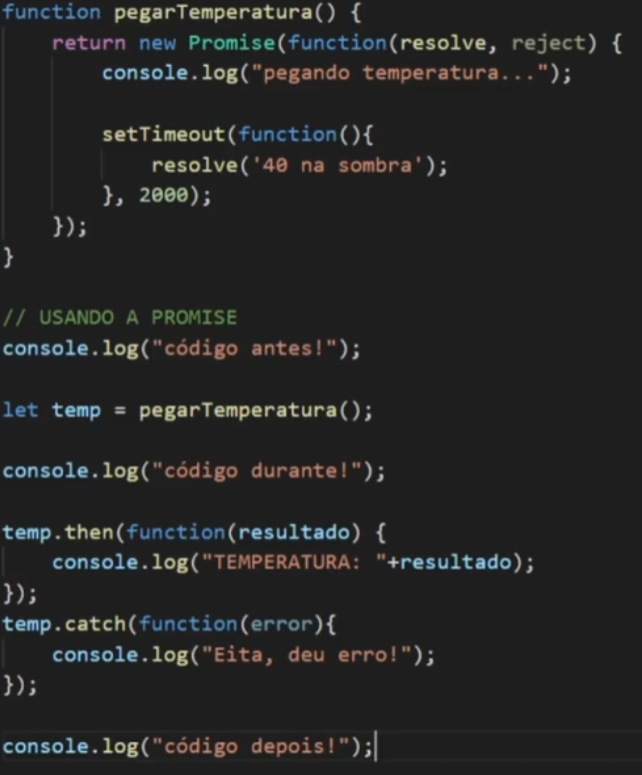
alert(“opa, tudo bem?”);

}

setTimout(alertar, 2000);

Ou seja, criei uma função e a usei em outra execução.

Promises

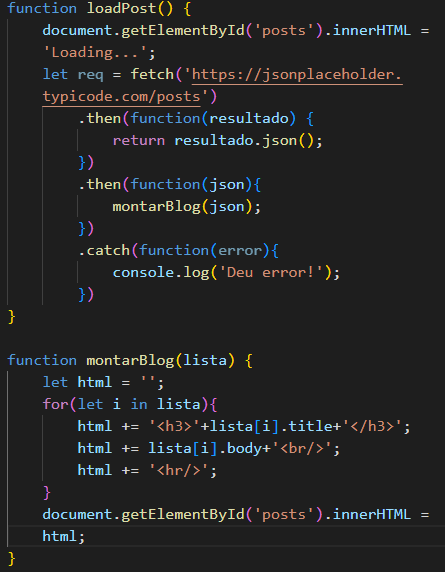
* É um resultado temporário ou uma promessa
* 

Requisição na Prática:

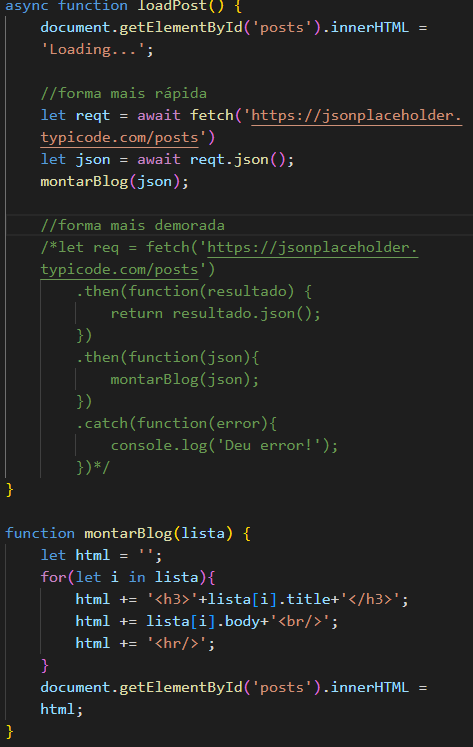
* Fetch 1 = ‘Naula014’



* Fetch 2 = Carregando os posts



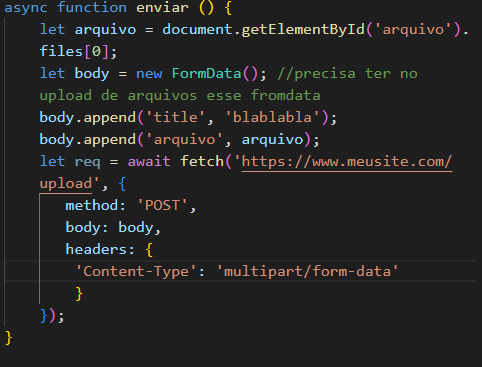
Async e Await: Diminui o código anterior



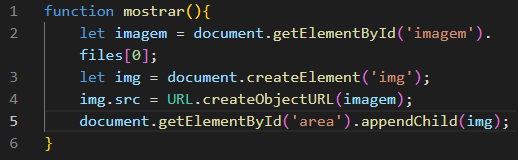
Fetch com POST: Poder usar o body nas requisições // Naula014



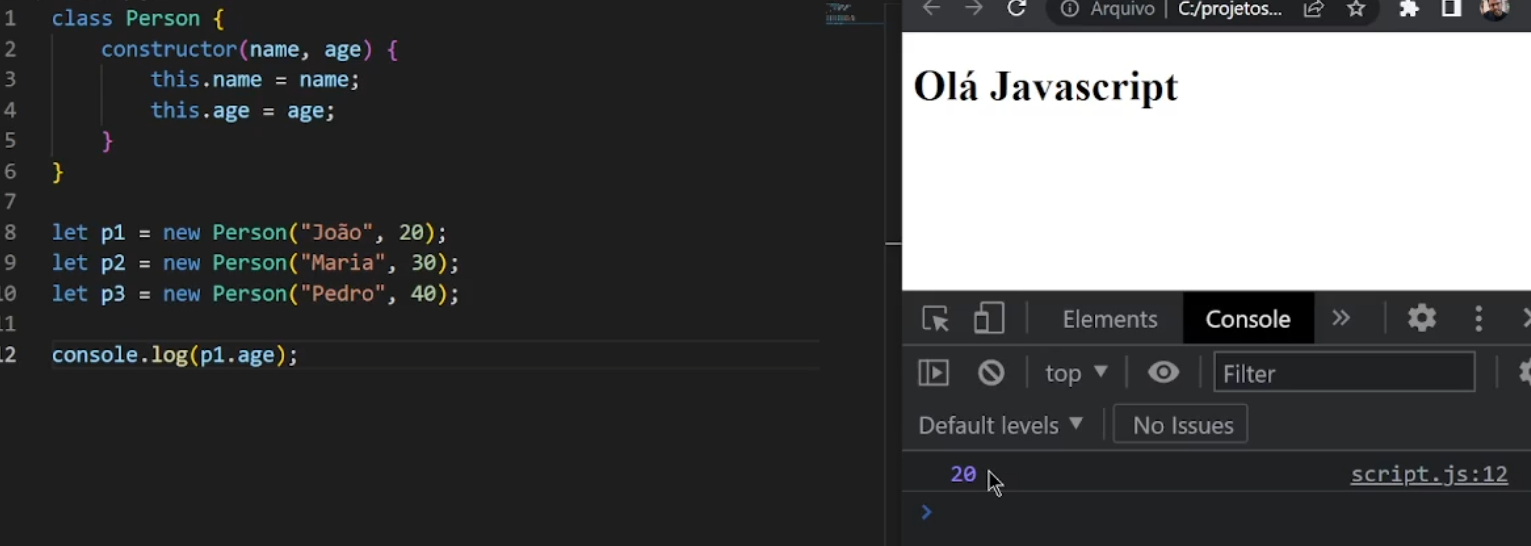
Upload de arquivos com JS: Aula Mod.8 #12 // Naula015



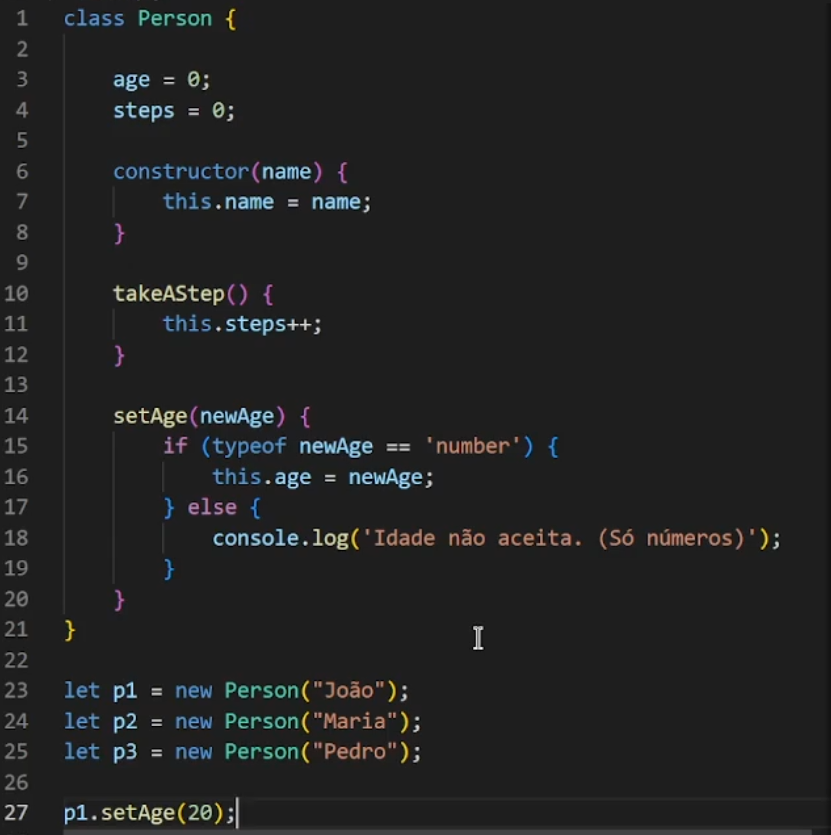
Thumbnails com JS: Disponível em “Naula016”



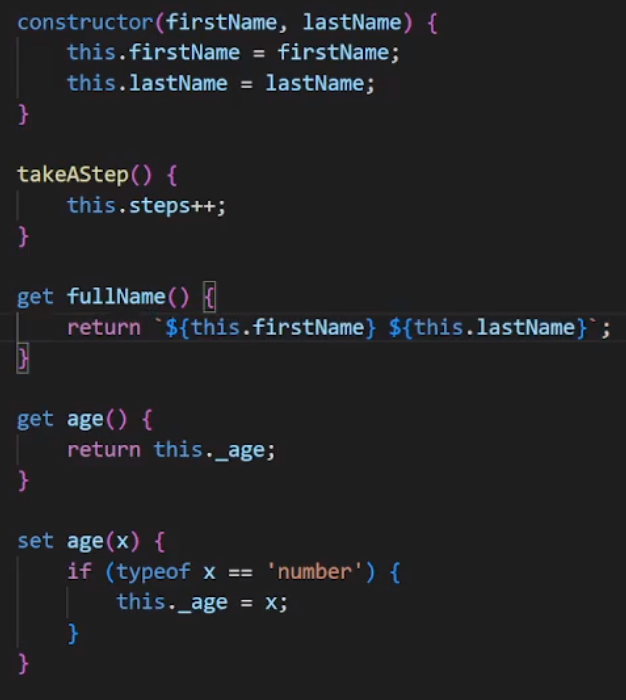
Orientação a Objetos:  
  
Fazendo um template e criando objetos para preenche-lo



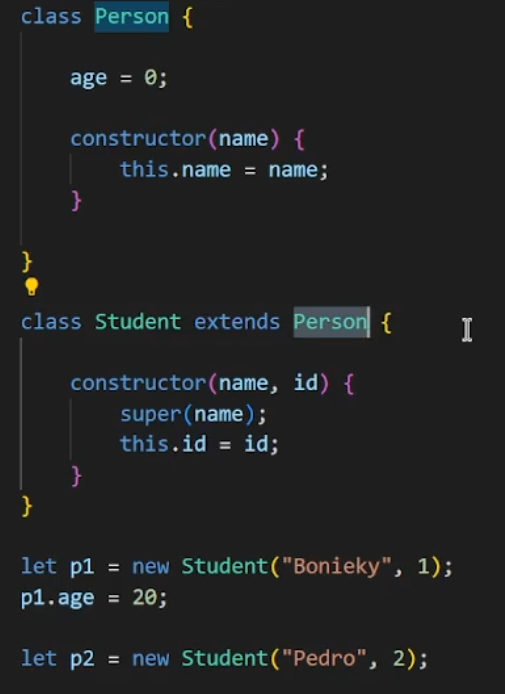
Sempre usa o **constructor** pra fazer a função, e o **this** pra se referir a variável.

Exemplos de ações:  


Get & Set  
Basicamente criando funções a serem executadas, get vai buscar uma informarção, e o set vai setar uma informação:



**Extends** quer dizer que o **Student** vai herdar tudo de **Person**



**Super** quer dizer que eu vou executar algo no **student** que já foi **setado** no **person**O que é estendido sempre sobrescreve o que foi extenso.  
Student sempre terá prioridade sobre Person.

**Requisições**