

JAVASCRIPT

JS

JS

JS

JOSÉ FABRÍCIO FIGUEIREDO

VARIÁVEIS

JS

var, let, const

Hoisting -> Variável sem valor dentro.

Immutability -> Não alterar valor da variável (const).

JS

JS

JOSÉ FABRÍCIO FIGUEIREDO

OBJETO E LISTA

JS

// Objeto

```
var obj = {nome:'José Fabrício', age:14}
```

- **.keys()**
- **.values()**

// Lista

```
var lista = new Array()
```

```
var lista = []
```

- **.pop()** -> Remove o ultimo.
- **.shift()** -> Remove o primeiro.
- **.splice()** -> Remove na escolha.
- **.unshift()** -> Adiciona no inicio.
- **.push()** -> Adiciona no final.
- **.concat(lista)** -> Uni duas listas.
- **.splice(index, remover, elemento)**

JS

JOSÉ FABRÍCIO FIGUEIREDO

LISTA

JS

- **.forEach((value, index) => {})**
- **.map(value => condição)**
- **.flat()** -> **[[0], [0]]** -> **[0, 0]**.
- **.flatMap(value => [operação])**
- **.find(value => condição)**
- **.findIndex(index => condição)**
- **.filter(value => condição)**
- **.indexOf()**
- **.some(value => condição)** -> Retorna um booleano verificando se pelo menos um item de um Array satisfaz a condição.
- **.every(value => condição)** -> Retorna um booleano verificando se todos os itens de um Array satisfazem a condição.

JS

JOSÉ FABRÍCIO FIGUEIREDO

LISTA

JS

- **.lastIndexOf()**
- **.includes()** -> Procura o valor.
- **.sort()**
- **.reverse()**
- **.join('-')**
- **.reduce((total, value) => operação, valor_inicial)**
- **.length**

JS

JOSÉ FABRÍCIO FIGUEIREDO

STRING

JS

- **.length**
- **.charAt()** -> Retorna o carácter.
- **.concat(txt2)** -> Junta duas string.
- **.includes()** -> Procura a palavra.
- **.indexOf()**
- **.repeat()** -> Repete a string.
- **.replace('antigo', 'novo')**
- **.search()** -> Procura a palavra.
- **.slice(inicio, fim)**
- **.split(separador, limite)**
- **.substr(inicio, fim)** -> Remova parte da string.
- **.toLocaleLowerCase()** -> Minúscula.
- **.toLocaleUpperCase()** -> Maiúscula.
- **.toLowerCase()** -> letra minúsculas.
- **.trim()** -> Remove espaço branco.

JS

JOSÉ FABRÍCIO FIGUEIREDO

CONDIÇÃO E LOOP

JS

- **if, else if, else**
- **switch case**
- **for, for-in(index), for-of(value)**
- **while, do-while**

JS

JS

JOSÉ FABRÍCIO FIGUEIREDO

OPERADORES

JS

// Ternários.

condição ? valor1: valor2;

// Lógico.

AND (&&), OR (||), NOT (!)

// Comparação.

==, ===, <=, >=, <, >, !=

// Binário (in)

JS

JOSÉ FABRÍCIO FIGUEIREDO

ACESSAR ELEMENTO

JS

document.querySelector('tag#id')

- **.getElementsById('id')**
- **.getElementByClassName('name')**
- **.getElementByName('name')**
- **.getElementByTagName('tag')**
- **.querySelectorAll('.class')**

JS

JOSÉ FABRÍCIO FIGUEIREDO

FUNCTION

JS

```
function nome(parâmetros) {  
  return  
}
```

// Currying

```
function nome(parâmetros) {  
  return function (parâmetros) {  
    return  
  }  
}
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

CRIANDO TAGS

HTML

JS

// Buscar elemento pai.

var elemento_pai = document.body

// Criar elemento.

var h1 = document.createElement('h1')

// Inserir.

elemento_pai.appendChild(h1)

// Criar o nó de texto.

**var texto = document.createTextNode("Um
título qualquer")**

// Anexar o nó de texto ao elemento h1.

h1.appendChild(texto)

JS

JOSÉ FABRÍCIO FIGUEIREDO

CONFIGURANDO ATRIBUTOS

JS

// Criar o atributo.

var att = document.createAttribute("class")

// Adicionar o valor no atributo criado.

att.value = "demo_class"

// Adicionar o atributos na tag

div.setAttributeNode(att)

tag.innerHTML = "

tag.innerText = "

JS

JOSÉ FABRÍCIO FIGUEIREDO

INSERIR E SUBSTITUIR ELEMENTOS HTML

JS

- **tag.childNodes** -> Elementos filhos.
- **tag.insertBefore(novo, referencia[0])**
- **tag.replaceChild(text_node, antiga_tag)**
- **tag.parentNode.nodeName** -> Elemento Mãe.
- **tag.remove()**
- **tag[0].getAttribute('atributo')**
- **tag.classList.toggle('class')** -> Substituir e remover class.

JS

JOSÉ FABRÍCIO FIGUEIREDO

CONTROLE DE TEMPO

JS

setTimeout(callback, tempo) -> A função será executada após o tempo.

setInterval(callback, tempo) -> A função é chamada no intervalo de tempo.

clearInterval(variavel_setInterval) -> Exclui a variável que tem o setInterval.

JS

JOSÉ FABRÍCIO FIGUEIREDO

TRATAMENTO DE ERRO

JS

```
try {  
    throw new Error('mensagem')  
} catch (err) {  
    console.log(err);  
} finally {  
    // Sempre executado  
}
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

DATA E HORA

JS

```
var res = new Date()
```

- **.getDate()** -> Dia do mês.
- **.getDay()** -> Dia da semana.
- **.getFullYear()** -> Ano.
- **.getHours()** -> Hora.
- **.getMilliseconds()** -> Milissegundos.
- **.getMinutes()** -> Minutos.
- **.getSeconds()** -> Segundos.
- **.getMonth()** -> Mês.
- **.toString()** -> Data em string.

JS

JOSÉ FABRÍCIO FIGUEIREDO

EVENTOS

JS

<tag onevento='ação(parâmetros)'\>

Parâmetros:

- **this** -> Responsável para acessar a tag utilizada.
- **event** -> Responsável enviar os eventos.

Evento:

- **mousedown** -> Pressionar o mouse.
- **mouseenter** -> Ponteiro do mouse.
- **mouseout** -> Mover o ponteiro para fora.
- **mousemove** -> Mover o ponteiro do mouse.
- **mouseup** -> soltar o clique do mouse.
- **click** -> Clicar.
- **canplaythrough** -> Video visto até o fim.

JS

JOSÉ FABRÍCIO FIGUEIREDO

EVENTOS

JS

- **change** -> Opção selecionada.
- **ended** -> Áudio visto até o fim.
- **error** -> Erro ao carregar a imagem.
- **focus** -> Campo de entrada ganhar foco.
- **pause** -> Vídeo for pausado.
- **play** -> Vídeo estiver começando.
- **select** -> Texto selecionado.
- **submit** -> Formulário enviado.
- **volumechange** -> Alterar o volume.
- **button** -> [1-meio do mouse], [0-esquerdo do mouse], [2-direito do mouse].
- **buttons** -> [3-direito e esquerdo], [4-meio], [7-todos], [5-esquerdo e meio], [6-direito e meio].

JS

JOSÉ FABRÍCIO FIGUEIREDO

EVENTOS

JS

- **pageX** -> Coordenada do mouse.
- **pageY** -> Coordenada do mouse.
- **load** -> Carregamento da pagina.

- **KeyDown** -> Tecla pressionada.
- **KeyUp** -> Tecla solta.
 - **event.which**

// Adicionar evento

```
var res = document.querySelector('')  
res.addEventListener('evento', função)
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

WINDOW

JS

window.parametro()

Parâmetros:

- **.open()** -> Abrir uma janela.
- **.close()** -> Fechar janela.
- **.innerWidth** -> Largura da janela.
- **.innerHeight** -> Altura da janela.
- **.screenX** -> posição da janela.
- **.screenY** -> posição da janela.
- **.scrollBy(x, y)** -> Rolar a janela.

JS

JOSÉ FABRÍCIO FIGUEIREDO

NAVIGATOR

JS

navigator.parametro();

Parâmetros:

- **.appName** -> Nome do navegador.
- **.appVersion** -> Versão do seu navegador.
- **.cookieEnabled** -> Cookis estão ativos.
- **.language** -> Idioma.
- **.onLine** -> Navegador está online.
- **.platform** -> Versão do navegador.
- **.userAgent** -> Agente do usuário.

JS

JOSÉ FABRÍCIO FIGUEIREDO

SCREEN

JS

screen.parametro()

Parâmetros:

- **.width** -> Largura da Tela.
- **.height** -> Altura da Tela.
- **.availWidth** -> Largura da janela.
- **.availHeight** -> Altura da janela.
- **.colorDepth** -> Paleta de cores em bits.
- **.pixelDepth** -> Resolução de cores.

JS

JOSÉ FABRÍCIO FIGUEIREDO

HISTORY

JS

window.history.parametro()

Parâmetros:

- **.back()** -> Volta uma pagina vista anteriormente.
- **.forward()** -> Ir para pagina seguinte.
- **.go()** -> Avançar e retornar. Ex: -2 e +2

JS

JOSÉ FABRÍCIO FIGUEIREDO

LOCATION

JS

location.parametro();

Parâmetros:

- **.hash** -> Âncora de um URL.
- **.hostname** -> Nome do host do URL.
- **.href** -> URL inteira da pagina.
- **.origin** -> Protocolo, host e número da porta.
- **.pathname** -> Caminho da URL.
- **.protocol** -> Protocolo da URL.
- **.search** -> A parte da string de consulta da URL.
- **.assing('url')** -> Carregar um novo documento.
- **.reload()**
- **.replace('url')**

JS

JOSÉ FABRÍCIO FIGUEIREDO

CLASS - EXTENDS

JS

```
class Teste {  
  constructor(valor) {  
    this.res = valor  
  }  
}
```

```
class Teste2 extends Teste {  
  constructor(valor, valor2) {  
    super(valor)  
    this.res = valor2  
  }  
}
```

```
var chamar = new Teste2("")
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

CLASS – SET GET

JS

```
class Nome_class {  
  constructor(valor) {  
    this.res = valor  
  }  
  
  set nome_função(valor) {  
    this.res = valor  
  }  
  
  get nome_função() {  
    return this.res  
  }  
}
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

CLASS – ENCAPSULAMENTO

JS

```
class Nome_class {  
  #res = ''  
  constructor(valor) {  
    this.#res = valor  
  }  
  get nome_função() {  
    return this.res  
  }  
  set nome_função(valor) {  
    this.#res = valor  
  }  
}
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

CLASS - STATIC

JS

```
class Nome_class {  
    static nome_função() {  
        // Método static  
    }  
}
```

Nome_class.nome_função()

JS

JOSÉ FABRÍCIO FIGUEIREDO

CLASS - PROTOTYPE

JS

// Altera a função existente.

Exemplo:

```
String.prototype.split = function() {  
    console.log('lxi...')  
}
```

```
console.log('123456'.split(''))
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

ARROW FUNCTION

JS

```
var res = (parâmetros) => {}
```

```
var obj = {  
  dado: function() {  
    this.log('value')  
  },  
  log: function(parâmetros) {}  
}
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

SPREAD

JS

```
var arr1 = [1, 2, 3]
```

```
var arr2 = [...arr1]
```

JS

JS

JOSÉ FABRÍCIO FIGUEIREDO

ENHANCED OBJECT LITERALS - P1

JS

// Valor de propriedade

var prop = 'Digital Innovation One'

var obj = {
 prop
}

// Adicionar função

function nome_função() {}

var obj = {
 nome_função
}

// Criar função

var obj {
 prop: **function**() {},
 prop() => {}
}

JS

JOSÉ FABRÍCIO FIGUEIREDO

ENHANCED OBJECT LITERALS - P2

JS

// Adicionar valor

```
var obj = {}
```

```
obj['prop'] = 'Digital Innovation One'
```

// Propriedade

```
var prop = 'prop'
```

```
var obj = {
```

```
    [prop]: 'Digital Innovation One'
```

```
}
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

DEFAULT FUNCTION ARGUMENTS

JS

```
function nome_fun(...args) {}
```

```
function nome_fun(x, y, z) {}
```

```
function nome_fun(x=0, y=nome_fun()) {}
```

JS

JS

JOSÉ FABRÍCIO FIGUEIREDO

REST OPERATORS E DESTRUCTURING

JS

// Parâmetros ilimitados (Rest Operators).

```
function nome_função(...args) {}
```

// Destructing.

```
var [res1, [res2]] = [valor1, [valor2]]
```

```
console.log(res1, res2)
```

```
>>> valor1, valor2
```

```
var obj {
```

```
  props: {
```

```
    res1: valor1,
```

```
    res2: [vaor2, valor3]
```

```
  }
```

```
}
```

```
var {props: {res1, res2: [cor1, cor2]}} = obj
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

PROMISES

JS

// São utilizados para verificar se o teste
houve Success ou Failed.

```
var myPromises = new Promise((resolved,  
reject) => {  
    if (condição) {  
        resolved('Sucess')  
    } else {  
        reject('Failed')  
    }  
}
```

```
myPromises.then((message) => {  
    console.log(message) // Success.  
}).catch((err) => {  
    console.log(err) // Failed.  
})
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

CALLBACK

JS

// Função assíncrona

JS

JS

JOSÉ FABRÍCIO FIGUEIREDO

SYMBOLS

JS

// São identificadores.

```
var unicoid = Symbol('name')
```

```
const iterable = {}
```

```
iterable[Symbol.iterator] = function* () {
```

```
  yield 1
```

```
  yield 2
```

```
  yield 3
```

```
}
```

```
console.log([...iterable])
```

```
>>> Array [1, 2, 3]
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

GENERATORS

JS

// * = Criar um iterado.

```
function* nome_função() {
```

```
  yield value
```

```
  // Quando chamado, a função para aqui e  
  continuara quando for chamado de novo.
```

```
}
```

```
const res = nome_função()
```

```
nome_função.next()
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

DESIGN PATTERNS

JS

// Factory -> Todas as funções que retorna um objeto, sem a necessidade de chama-las com o new.

Exemplo:

```
function nome_função(valor) {  
  return {  
    key: valor  
    ...valor  
  }  
}
```

```
var res = acao({key:value})
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

DESIGN PATTERNS

JS

// Singleton -> O objeto desse pattern é criar uma única instância de uma função construtora e retorná-la toda vez em que for necessária utilizá-la.

Exemplo:

```
function acao() {  
    if (!MyApp.instance) {  
        MyApp.instance = this  
    }  
    return MyApp.instance  
}
```

```
const p = acao.call({name: 'José'})  
// {name: 'José'}  
const p2 = acao.call({name: 'Javascript'})  
// {name: 'Javascript'}
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

DESIGN PATTERNS

JS

// Decorator -> Recebe outra função como parâmetro e estende o seu comportamento sem modificá-la explicitamente.

Exemplo:

```
let logendin = false  
function callIfAuthenticated(fn) {  
  return !!logendin && fn()  
}  
function soma(a, b) {  
  return a + b  
}  
console.log(callIfAuthenticated(() => soma(2,  
3)))
```

JS

JOSÉ FABRÍCIO FIGUEIREDO

DESIGN PATTERNS

JS

// Module -> É um pattern que possibilita organizarmos melhor o nosso código, sem a necessidade de expor variáveis globais.

Exemplo:

```
class Person {  
  constructor(name) {  
    this.name = name  
  }  
}  
export default Person
```

// Utilizar Person

```
import Person from '../Design_patterns/ex5'
```

JS

JOSÉ FABRÍCIO FIGUEIREDO