**Criando elementos html pelo JS**

**constdiv** = **document**.**createElement(“div”)**

**div.innerHTML = “//Conteúdo ”**

**div.setAttribute(“class”, “//nome”)  
container.appendChild(div) //Class dada no html onde se quer app o conteúdo**

**Métodos JavaScript**

**Métodos de Manipulação de Arrays:**

* array. **push** (“x”) // Insere um elemento “x” no final
* array. **pop** (“x”) // Retira um elemento “x” do final
* array. **unshift** (“x”) // Insere um elemento “x” no início
* array. **shift** (“x”) // Retira um elemento “x” do início
* array. **map** ((x , i) => { //App uma função a cada elemento do array e gera uma novo array com os elementos operados (i = índice da posição do elemento “x”)

//Função

})

* array. **indexOf** (“x”) // Retorna à posição do elemento. Se o elemento não existir retorna "-1".

**Métodos de Manipulação de Strings:**

* string. **charAt** (0) // Retorna a letra da posição[0]
* string. **charCodeAt** (0) // Retorna o código (do teclado) relacionado ao caractere na posição[0]
* string. **concat** (sting2) // Concatena a “string” com a “string2”
* string. **indexOf** (“x”) // Retorna a posição do primeiro caractere “x”. Se não existir retorna “-1”
* string. **lastIndexOf** (“x”) // Retorna a posição do último caractere “x”. Se não existir retorna “-1”
* string. **localeCompare** (string2) //Compara as strings e verifica se são iguais. Se True, retorna “0”, se False, retorna “1” caso “string > string2” ou retorna “-1” caso “string < string2”
* string. **replace** (“x”, ”y”) // Substitui o primeiro caractere “x” por “y” (Também funciona para palavras inteiras)
* string. **search** (“Pinho”) // Encontra a posição do primeiro caractere da palavra passada no parâmetro.
* string. **slice** (6 , 11) // Retorna o recorte da string (start, end)
* string. **split** (“ “) // Retorna o um array recortando a string onde há um “ “ (espaço vazio)
* string. **substring** (0 , 5) //Parecido com o “slice”, retorna o recorte da string (start, end)
* string. **toUpperCase( )** // Converte tudo para letras Maiúsculas
* string. **toLowerCase( )** // Converte tudo para letras Minúsculas
* string. **startsWith(**“Pinho”) // Verifica se a string começa com o parâmetro passado
* string. **endsWith(**“Pinho”) // Verifica se a string termina com o parâmetro passado
* string. **includes(**“X”) // Verifica se a string possui o parâmetro passado em alguma parte dela
* string. **repeat(**4) // Indica quantas vezes repetir a string

**Métodos De Data:**

* getDate() // Pega o Dia do Mês
* getDay() // Pega o Dia da semana (número)
* getFullYear() // Pega o Ano com 4 dígitos
* getHours() // Pega as Horas
* getMinutes() // Pega os Minutos
* getSeconds() // Pega os Segundos
* getMilliseconds() // Pega os Milissegundos
* getTime()// Timestamp (milissegundos desde 1 de janeiro de 1970, 00:00:00 UTC)
* Date.now()// Timestamp (milissegundos desde 1 de janeiro de 1970, 00:00:00 UTC)
* getTimezoneOffset() // Timezone da localidade

**Métodos da Classe Elements HTML:**

**el.getBoundingClientRect() //Retorna as propriedades de um elemento HTML em um objeto DOM (*posição, altura, largura etc.*)**

**el.acessKey=”b” //Propriedade que permite definir uma tecla de atalho (“Alt+b” irá ativar o comando)**

**Métodos de Validação do DOM:**

**el.checkValidity() //Checa de o campo for preenchido corretamente ou não**

**el.setCustomValidity(“Mensagem aqui”) //Permite customizar a mensagem**

**Propriedades de Validação do DOM:**

**el.Validity**

**el.validationMessage**

**Propriedades de Validação:**

**customError:** true, se uma mensagem de validação personalizada for definida.

**patternMismatch:** true, se o valor de um elemento não corresponder ao seu atributo padrão.

**rangeOverflow:** true, se o valor de um elemento for maior que seu atributo max.

**rangeUnderflow**: true, se o valor se um elemento for menor que seu atributo min.

**stepMismatch:** true, se o valor de um elemento for inválido por seu atributo step.

**tooLong:** true, se o valor de um elemento exceder seu atributo maxLength.

**typeMismatch:** true, se o valor de um elemento for inválido por seu atributo type.

**valueMissing:** true, se um elemento (com um atributo obrigatório) não tiver valor.

**valid:** true, se o valor de um elemento for válido.

**Redirecionamento de Páginas**

window.location = “aula01.html” //redireciona a página para a aula01

window.location = “https://www.google.com.br” //redireciona a página do google

**Métodos Location:**

window.location.**replace**(“https://www.google.com.br”) //substitui a URL corrente pela URLdo google a excluindo do histórico (Impossibilitando a volta)

window.location.**assign**(“https://www.google.com.br”) //substitui a URL corrente pela URL do google, porém sem excluir do histórico (Possibilitando a volta)

window.location.**reload**() //Recarrega a URL corrente

**Window History:**

window.history.**back**() //Volta para a URL anterior gravada no histórico

window.history.**forward**() //Avança para a próxima URL gravada no histórico

window.history.**go**(1) //Avança ou volta para aURL indicada por um number no parâmetro

**Caixas de diálogo**

**alert**(“Olá como está?”) //Mostra uma caixa de msg com o texto passado

**confirm**(“Deseja continuar?”) //Mostra uma caixa de msg com o texto passado e retorna um booleano com ‘true’ ou ‘false’

**prompt**(“Digite seu nome.”) //Mostra uma caixa de msg com o texto passado e retorna uma string ou um null

**Conteúdo para Impressão**

window.**print**() // vai gerar um print da tela (como um todo) para impressão

//Para imprimir apenas o conteúdo importante, vale criar um media query separado apenas para impressão

**Objetos Literais**

constcomputador = {    //As {} já indicam que é um objeto

    cpu:"i9",

    ram:"64gb",         //As propriedades devem ser separadas por vírgula

    hd:"2tb"

}

constcomputadores = [      // Array de Objetos Literais

    {

        cpu:"i9",

        ram:"64gb",

        hd:"2tb"

    },

    {

        cpu:"i7",

        ram:"32gb",

        hd:"2tb"

    },

    {

        cpu:"i5",

        ram:"16gb",

        hd:"1tb"

    }

]

constcomputador = {

    cpu:"i9",

    ram:"64gb",

    hd:"2tb",

    info:function(){ //Passando métodos

        console.log(`CPU: ${this.cpu}`)

        console.log(`RAM: ${this.ram}`)

        console.log(`HD: ${this.hd}`)

    }

}

computador.info() //Chama o método

//Para Adicionar propriedades temos duas formas:

    computador["monitor"]="22pol"

    computador.placaVideo="rtx"

**Clonando um Objeto Literal:**

//Clonando um objeto literal

constcomputador = {

    cpu:"i9",

    ram:"64gb",

    hd:"2tb",

    info:function(){

        console.log(`CPU: ${this.cpu}`)

        console.log(`RAM: ${this.ram}`)

        console.log(`HD: ${this.hd}`)

    }

}

constc1 = Object.assign({}, computador)    //C1 passa a ser um objeto exatamente igual a computador

//Mesclando vários Objetos em um só

consto1={obj1:"1"}

consto2={obj2:"2"}

consto3={obj3:"3"}

consto4 = Object.assign(o1, o2, o3)    //o4 será a junção de o1, o2 e o3

**Módulos**

Resumidamente os **módulos em JavaScript** são uma forma de compartilhar informações entre arquivos, dessa forma, todos os arquivos que realizarem a importação desse **módulo** poderão utilizar e usufruir de suas funções ou informações que foram expostas. Dessa maneira, evitamos ter que ficar carregando N arquivos .

**Rodando no Servidor:**

//Informar no arquivo HTML o 'type="module"'

    <scriptsrc="modulo01.js"type="module"></script>

**//Arquivo.js 2**

constcursos = []

constcarros = []

export {cursos, carros} //Exporta os Arrays para o arquivo.js 1

**//Arquivo.js 1**

import { cursos } from"./modulo02.js";//Recebe os Arrays do Arquivo.js 2

import { carros } from"./modulo02.js";

**Módulo em Classes:**

**Exemplo:**

**//Arquivo.js 3**

classCursos{

    staticcursos = ["JavaScript","HTML","CSS"] //'Static' permite não precisar instanciar a classe para poder utilizá-la

    constructor(){

}

    staticgetAllCursos=()=>{   //Método que retorna o Array cursos

        returnthis.cursos

    }

    staticgetCurso=(i\_curso)=>{    //Método que retorna o curso do índice passado no parâmetro

        returnthis.cursos[i\_curso]

    }

    staticaddCurso=(newCurso)=>{   //Método que add um curso novo ao Array cursos

        this.cursos.push(newCurso)

    }

    staticRemoveCursos=()=>{       //Método que limpa todo o Array cursos

        this.cursos = []

    }

}

exportdefaultCursos//'default' classifica como padrão e permite alterar o nome da classe no arquivo que irá importar (Não usa as chaves)

**//Arquivo.js 1**

//Como a classe está sendo exportada como ‘default’, podemos alterar o nome da classe 'cursos' para 'c'

importcfrom"./modulo03.js"

  //Chama os métodos

    c.RemoveCursos()

    c.addCurso("Photoshop")

    console.log(c.getCurso(0))

    console.log(c.getAllCursos())

**Funções**

**Função Symbol:**

//A função Symbol retorna um objeto do tipo Symbol que tem um identificador único. (Não se pode usar esse identificador no DOM)

consts1 = Symbol()

consts2 = Symbol()     //(s1==s2) Irá retornar 'false'

consts3 = Symbol.for("Carlos")//Cria um identificador no registro global

consts4 = Symbol.for("Carlos") //Esse identificador pode ser dado tbm a outro Symbol

                        //(s3==s2) Agora irá retornar 'True'

// 'Symbol.keyFor()' Retorna a string do identificador ou um undefined (trabalha apenas com os registros globais)

console.log(Symbol.keyFor(s1))     //Retornará 'undefined'

console.log(Symbol.keyFor(s3))     //Retornará 'Carlos'