**ARROW FUNCTION**

// Utilizando arrow function (na arrow function pode omitir o return, pois é implicito e por ser uma linha, podemos remover as chaves)

const maxNumero2 = (*x*, *y*) => *x* > *y* ? `${*x*} > ${*y*}` : `${*y*} > ${*x*}`;

Cria uma variável com que recebe os valores da função e com o símbolo => aponta para a função que ele trata

**TRATANDO E LANÇANDO ERROS**

Utilizamos o try catch para realizar o tratamento de erros em nosso código, podendo também levantar erros com o throw.

function soma (*x*, *y*) {

    if (typeof *x* !== 'number' || typeof *y* !== 'number') {

        throw **new** *TypeError*('x e y precisam ser números');

    }

    return *x* + *y*;

}

try {

    console.log(soma(1, 2));

    console.log(soma('1', 2));

} catch(error) {

    // console.log(error)

    console.log('Alguma coisa mais amigavel para o usuário')

}

Junto com o try e catch pode ser usado tbm o finally, que sempre será executada, independente de ter erro ou não:

try {

    // É executado quando não há erros

    console.log('Abri um arquivo')

    console.log('Manipulei um arquivo e gerou erro')

} catch (e) {

    // É executado quando há erros

    console.log('Tratando o erro')

} finally {

    // Sempre é executado

    console.log('FINALLY: Fechei o arquivo')

}

**REMOVENDO ESPAÇOS A MAIS:**

trim() -> remove os espaços em branco antes e depois da string

**FUNÇÕES**

Em JavaScript, existem algumas preculiaridades envolvendo os argumentos das funções.

Primeiro, o JS aceita que uma função que foi declarada sem receber nenhum parametro receba valores enviados em sua chamada, sem causar nenhum erro.

Todas os valores enviados são adicionados a um objeto de nome ‘arguments’, que pode ser utilizado dentro da função. Exemplo:

// IMPORTANTE: Em funções decladas com a palavra functions, os valores enviados na chamada da função são todos colocados no objeto 'arguments'

function funcao() {

    let total = 0;

    for (let argumento of *arguments*) {

        total += argumento;

    }

    console.log(total);

}

funcao(1, 2, 3, 4, 5, 6);

*\*IMPORTANTE: isso só acontece em funções que são declaradas com o function antes delas. Em arrow functions por exemplo não temos a mesma propriedade.*

Outra coisa é que quando é enviado menos valores do que parametros declarados na função, o JS também não retorna um erro, mas sim aplica o valor ‘undefined’ para os parametros que não receberam valores. Exemplo:

function funcao(*a*, *b*, *c*, *d*, *e*, *f*) {

    console.log(*a*, *b*, *c*, *d*, *e*, *f*);

}

funcao(1, 2, 3);

>>> RETORNO:

1 2 3 undefined undefined undefined

**Adicionando todos os valores restantes a uma variavel com rest:**

function conta(*operador*, *acumulador*, ...*numeros*) {

    for (let numero of *numeros*) {

        if (*operador* === '+') *acumulador* += numero;

        if (*operador* === '-') *acumulador* -= numero;

        if (*operador* === '/') *acumulador* /= numero;

        if (*operador* === '\*') *acumulador* \*= numero;

    }

    console.log(*acumulador*)

}

conta('/', 1, 20, 30, 40, 50);

**\*\*IMPORTANTE:**

shift()-> remove o primeiro ELEMENTO de uma array e retorna esse ELEMENTO