# Antes de mais nada...

Irei recomendar mais plugins para o VS Code:

* Dracula Official
* Material Icon Theme

# Operadores de Comparação

* > : maior que
* < : menor que
* >= : maior ou igual que
* <= : menor ou igual que
* == : igualdade (valor)
* === : restritamente igual (valor e tipo)
* != : diferente (valor)
* !=== : restritamente diferente (valor e tipo)

# Operadores Lógicos

* && : E (and) – todas tem que ser verdadeiras para retornar true
* || : OU (or) – todas tem que ser falsas para retornar false
* ! : NÃO (not) – inverte os valores de true e false

# Short-Circuit

Quando usamos o operador &&, automaticamente é retornado o valor referente a false. Podemos usar isso para parar a execução de algo não desejado.

Obs: caso tudo seja true, retornará o último valor da comparação

function ola(){

    return 'ola'

}

let vaiExecutar = false

console.log(vaiExecutar && ola()) //resultado: false

Podemos fazer essa avaliação com || também:

Obs: caso tudo seja true, retornará o primeiro valor da comparação

function ola(){

    return 'ola'

}

let vaiExecutar = true

console.log(vaiExecutar || ola()) //resultado: true

# If, else if, else

If é uma estrutura condicional que, caso aconteça nossa condição, será executado nosso código:

function se(){

    const lampada = 'funcionando'

    let resultado = 0

    if (lampada === 'queimada' || lampada === 'sem lâmpada') {

        resultado = 'trocar lâmpada'

    }else if(lampada === 'funcionando'){

        resultado = 'lâmpada está funcionando'

    }else{

        resultado = 'não há lâmpada'

    }

    return resultado

}

console.log(se()) // 'lâmpada está funcionando'

# Operador Ternário

O operador ternário faz a mesma coisa que a função *if* e é representado pelo operador *?*

function ternario(){

    const nota = 5

/\*\*\*\*\*se nota >=6, então 'Aprovado', se não, 'Reprovado'\*/

    return nota >= 6 ? 'Aprovado' : 'Reprovado'

}

# Objeto Date

O objeto date possui funções para manipularmos o tempo, horas e etc.

function date(){

    const data = **new** *Date*()

    return data.toString()//retorna a data e hora do momento de compilação do código

}

# Switch/Case

Um switch é cheio de opções e, caso receba um valor, uma função específica será executada.

function calculadora(*num1* = 0, *num2* = 0, *menu* = 0){

    let resultado = 0

    switch(*menu*){

        case 1:

            resultado = 'Adição: '

            resultado += *num1* + *num2*

            break

        case 2:

            resultado = 'Subtração: '

            resultado += *num1* - *num2*

            break

        case 3:

            resultado = 'Multiplicação: '

            resultado += *num1* \* *num2*

            break

        case 4:

            resultado = 'Divisão: '

            resultado += *num1* / *num2*

            break

        default:

            resultado = `Selecione uma opção de 1 a 4`

    }

    return resultado

}

# Atribuição via desestruturação (Arrays)

Atribui valores, a partir de um array, de uma maneira diferente

function atribuicaoArrays(){

    const numeros = [1,2,3,4]

    const [primeiro, segundo] = numeros

    //             1         2

    console.log(primeiro, segundo)

}

# Atribuição via desestruturação (Objetos)

Atribui valores, a partir de um objeto, de uma maneira diferente

function atribuicaoObjeto(){

    const pessoa = {

        nome: 'silas',

        idade: 21

    }

    const {nome, idade} = pessoa

    console.log(nome,idade)

}

# For - Estrutura de repetição

function lacofor(*x*){

    for(let i = 0; i<=*x*; i++){

        console.log(i)

    }

}

# For in - Estrutura de repetição

Retorna os elementos índices da variável, bom quando não sabemos o tamanho do que será percorrido.

function forin(){

    let frutas = ['maça', 'uva', 'abacaxi']

    for(let i in frutas){

        console.log(frutas[i])

    }

}

# For of - Estrutura de repetição

Retorna os valores da variável, bom quando não sabemos o tamanho do que será percorrido.

function forof(){

    let nome = 'Silas'

    for(let i of nome){

        console.log(i)

    }

}

# While e Do While - Estrutura de repetição

Uma estrutura que será executada enquanto a condição for verdadeira

function dowhile(){

    function random(*min*,*max*){

        const r = Math.random() \* (*max*-*min*) + *min*

        return Math.floor(r)

    }

    let min = 1

    let max = 10

    let rand = random(min,max)

    while(rand !== 9){

        rand = random(min,max)

        console.log(rand)

    }

}

# Break e Continue

Continue, pula a repetição quando encontrado. Já o break, ao ser ativado, sai do laço de repetição

function breakContinue(){

    const numeros = [1,2,3,4,5]

    for ( let numero of numeros){

        if (numero === 2){

            console.log('Pulei o 2')

            continue

        }

        console.log(numero)

        if(numero === 4){

            console.log('4 encontrado, saindo...')

            break

        }

    }

}

# Tratando e lançando erros (try, catch, throw)

Quando há um código propenso a erros, você precisa inserir um *try* que tenta executar um código que, ao dar erro, executa outro bloco de código, através do comando *catch.*

**IMPORTANTE:** exibir erros ou falhas para o usuário final, pode ser muito perigoso, pela exposição de código que pode gerar. É recomendado tratar o erro da forma que lhe for mais conveniente.

Podemos também, lançar um erro e aparecer apenas uma mensagem que escolhemos.

function soma(*num1*, *num2*){

    if( typeof *num1* !== 'number' || typeof *num2* !== 'number'){

        throw ('num1 ou num2 não são números')

    }

    return *num1*+*num2*

}

try{

    console.log(soma(1,2))

    console.log(soma('1',2))

}catch(error){

    console.log(error)

}

# Tratando e lançando erros (try, catch, finally)

Temos também, o bloco *finally*, que, ao final do tratamento de erro, sempre será executado. Deve ser usado quando for viável ao programador (como um fechamento de arquivo ao final da manipulação, ...).

function soma(*num1*, *num2*){

    if( typeof *num1* !== 'number' || typeof *num2* !== 'number'){

        throw ('num1 ou num2 não são números')

    }

    return *num1*+*num2*

}

try{

    console.log(soma(1,2))

    console.log(soma('1',2))

}catch(error){

    console.log(error)

} finally{

    console.log('\*\*fim do tratamento de erros\*\*')

}

# setInterval e setTimeout

A função *setInterval* pode programar uma função para ficar sendo executada sempre durante um intervalo de milissegundos

Já a função *setTimeout*, determina um tempo para parar a execução do *setInterval*.

function setIntervalSetTimeout(){

    function falaOi(){

        return 'oi'

    }

    const mensagem = setInterval(function(){

        console.log(falaOi())

    }, 1000)//executa falaoi() a cada 1seg

    setTimeout(function(){

        clearInterval(mensagem)

    }, 3000) //para o setInterval após 3seg

    setTimeout(function(){

        console.log('chega de ola')

    },5000)

}