**Java Script**

O JavaScript é uma linguagem de programação de alto nível, que integra o desenvolvimento de apps e páginas web.

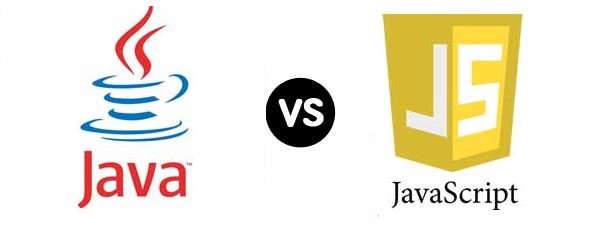
(Alto nível, se assemelha a linguagem humana. Baixo Nivel, bytecode)

É uma das linguagens de programação mais usadas no mundo do desenvolvimento.

**Como Surgiui?**

Em 1995 o programador Brendan Eich que trabalhava na Netscape, implementou uma linguagem de programação chamada Mocha, depois o nome mudou para LiveScript, mas na época a linguagem Java era popular, então mudaram LiveScript para JavaScript como jogada de marketing.

Isto causa confusão até hoje, coisa entre JavaScript e Java, porque muitos pensam que é a mesma, embora sejam coisas diferentes. Isto é culpa do mrketing.



**Qual é a diferença entre o JavaScript e o Java?**

A linguagem de programação JavaScript, desenvolvida pela Netscape, Inc., não faz parte da plataforma Java.

O JavaScript não cria applets ou aplicações independentes. Na sua forma mais comum, o JavaScript fica embutido nos documentos HTML e pode fornecer níveis de interatividade para páginas Web que não são acessíveis com um HTML simples.

Diferenças-chave entre o Java e o JavaScript:

* Java é uma linguagem de programação OOP, ao passo que Java Script é uma linguagem de scripts OOP.
* Java cria aplicações executadas em uma máquina virtual ou em um browser, ao passo que o código JavaScript é executado apenas em um browser.
* O código Java precisa ser compilado, ao passo que os códigos JavaScript estão totalmente em texto.
* Eles requerem plug-ins diferentes.

**Característica da linguagem**

O JavaScript, como o próprio nome sugere, é uma linguagem de scripting.

Uma linguagem de scripting é comumente definida como uma linguagem de programação que permite ao programador controlar uma ou mais aplicações de terceiros.

No caso do JavaScript, podemos controlar alguns comportamentos dos navegadores através de trechos de código que são enviados na página HTML.

Outra característica comum nas linguagens de scripting é que normalmente elas são linguagens interpretadas, ou seja, não dependem de compilação para serem executadas.

Essa característica é presente no JavaScript: o código é interpretado e executado conforme é lido pelo navegador, linha a linha, assim como o HTML.

O JavaScript também possui grande tolerância a erros, uma vez que conversões automáticas são realizadas durante operações.

Como será visto no decorrer das explicações, nem sempre essas conversões resultam em algo esperado, o que pode ser fonte de muitos bugs, caso não conheçamos bem esse mecanismo. O script do programador é enviado com o HTML para o navegador, mas como o navegador saberá diferenciar o script de um código html? Para que essa diferenciação seja possível, é necessário envolver o script dentro da tag <script></script>.

**Para quê serve?**

Para criar scripts dinâmicos que realizam a interação de apps ou páginas web.

Imagina que o HTML é a estrutura, o CSS é o design, o JavaScript é a parte interativa do desenvolvimento.

**Onde Usar?**

Podemos usar o JavaScript de várias formas, seu uso mais comum se dá no desenvolvimento de apps e páginas web, juntamente com as linguagens HTML e CSS.

**Links**

Glossário – developer.mozilla.org/pt-BR/docs/Glossary/JavaScript

O que é o JavaScript –

developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First\_steps/W

hat\_is\_JavaScript

**Configurando e Desenvolvendo**

Vamos utiliza o VSCode e extensão Live server

No vsCode, clicar em extensões e procurar LiveServer e instalar.

Criar uma para pasta chamda aula 1

Criar um arquivo chamado index.html

! enter o vcs cde monta a pagina html

Trocar o title e acrescentar o body.

Incluir uma Tag <scritp> eu utilizar um dos métodos do java script: alert.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Meu Primeiro Scripit</title>

</head>

<body>

    <h1>Meu primeiro Script</h1>

    <script>

        alert("Bem vindos!")

    </script>

</body>

</html>

Para executar: Botão direito sobre o index.html, abrir com live server. Vai abrir uma aba no navegador e executar.

**Boas Práticas**

1 - Externar o arquivo, criar um arquivo com extensão .js para o nosso scripti

<body>

    <h1>Meu primeiro Script</h1>

    <script src="script.js"></script>

</body>

Coloca no CTRL e clica em sima do nome, e o vscode criara o arquivo.

2 - carregar o script ao final do body. Primeiro todo o html e depois a parte funcional.

**Como é a tipagem em JavaScript?**

A tipagem funciona como uma regra de uso de dados, quanto mais forte for a tipagem, mais obrigatório é a declaração do tipo de dado.

A tipagem em JavaScript é fraca, a declaração dos dados acontece de modo dinâmico.

Ex.: Ao criarmos uma variável com valor entre aspas (“valor”) o JavaScript já converte o dado para o tipo String.

Ex.: var numero = 1;

// o JavaScript já converte o valor 1 para o tipo Number.

**Tipos primitivos**

As variáveis em JavaScript podem guardar tipos de dados que chamamos de tipos primitivos.

Variáveis podem guardar valores dos tipos: Boolean; null; undefined; Number; String; Array; Object; Function.

Exemplos:

> var resultado = 102 / 17;

undefined

No exemplo acima, guardamos o resultado de 102 / 17 na variável resultado. O resultado de criar uma variável é sempre undefined. Para obter o valor que guardamos nela ou mudar o seu valor, podemos fazer o seguinte:

> resultado 6

> resultado = resultado + 10

16

> resultado 16

Também podemos alterar o valor de uma variável usando as operações básicas com uma sintaxe bem compacta:

> var idade = 10; // undefined

> idade += 10; // idade vale 20

> idade -= 5; // idade vale 15

> idade /= 3; // idade vale 5

> idade \*= 10; // idade vale 50

**Number**

Com esse tipo de dados é possível executar todas as operações que vimos anteriormente:

var pi = 3.14159;

var raio = 20;

var perimetro = 2 \* pi \* raio

**String**

Não são apenas números que podemos salvar numa variável. O JavaScript tem vários tipos de dados. Uma string em JavaScript é utilizada para armazenar trechos de texto:

var empresa = "Funtec";

Para exibirmos o valor da variável empresa fora do console, podemos executar o seguinte comando: alert(empresa);

O comando alert serve para criação de popups com algum conteúdo de texto que colocarmos dentro dos parênteses. O que acontece com o seguinte código?

var numero = 30;

alert(numero)

O número 30 é exibido sem problemas dentro do popup. O que acontece é que qualquer variável pode ser usada no alert. O JavaScript não irá diferenciar o tipo de dados que está armazenado numa variável, e se necessário, tentará converter o dado para o tipo desejado.

(É possível omitir o ponto e vírgula no final de cada declaração. A omissão de ponto e vírgula funciona no JavaScript devido ao mecanismo chamado automatic semicolon insertion (ASI).)

Criar o index

Criar o script.js

//tipos primitivos

//typeof mostra o tipo da variável

//boleanos

var vOuf = false;

console.log(typeof (vOuf) );

//number

var umNumero = 1;

console.log(typeof(umNumero));

//string

var nome = "shirlei";

console.log(typeof(nome));

Abrir com live server

, no browser, F12 ou inspecionar, console

**Declaração de variáveis**

Existem 3 modos de declarar as variáveis em JavaScript:

var – (genérica) escopo global e local, pode ter seu valor alterado, se não tiver um valor inicial será tratada como null;

Ex:

var variavel = 'Shirlei';

variavel = 'Gabriel';

console.log(variavel);

let – escopo local de bloco, pode ter seu valor alterado, se não tiver um valor inicial será tratada como null;

Por convenção usamos em escopo local.

let variavel2 = 'shirlei';

//variavel2 = 'Gabriel';

console.log(variavel2);

const – escopo local de bloco, somente leitura, o valor inicial é obrigatório e não pode ser alterado.

const constante;

console.log(constante);

//se mandarmos rodar assim retornará um erro

const constante = 'shirlei';

console.log(constante);

//se mudarmos o valor, ela irá retornar um erro.

**Escopo**

O escopo em JavaScript define a limitação e visibilidade de um bloco de código.

Escopo global – quando a variável é declarada fora de qualquer bloco, sua visibilidade fica disponível em todo o código.

Escopo local – quando a variável é declarada dentro de um bloco, sua visibilidade pode ficar disponível ou não.

Ex.

var escopoGlobal = 'global';

console.log(escopoGlobal);

function escopoLocal() {

    let escopoLocalInterno = 'local';

    }

    Console.log(escopoLocalInterno);

// se rodarmos irá mostrar a variável local, mas retornara um erro dizendo que escopo localInterno não foi definido.

Porem foi definido, só não esta sendo visto, pois como esta dentro de uma função, é local.

Para resolver:

var escopoGlobal = 'global';

console.log(escopoGlobal);

function escopoLocal() {

    let escopoLocalInterno = 'local';

    console.log(escopoLocalInterno);

    }

escopoLocal();

**Regras de Uso de Variáveis**

• Iniciar com letras, underscore \_ ou cifrão $; Não iniciar com número.

Ex.: var 1nome ❌

var nome || var \_nome ✔

• Não usar espaços (use o camelCase ou \_);

Ex.: var nome completo ❌

var nomeCompleto || var nome\_completo✔

• Não usar palavras reservadas;

Ex.: var function ❌

• Declarar variáveis no topo do bloco de código.

**Atribuição**

O sinal de igualdade “=“ em JavaScript, significa atribuição.

Ex.:

como declarar: var nome = “meu nome”;

como ler: variável nome recebe o valor meunome;

**Comparação**

Para fazermos uma comparação de valores em JavaScript usamos

“==“.

Ex.:

como declarar: "0" == 0;

como ler: “0” tem o valor igual a 0?

// nesse caso retorna true

**Comparação idêntica**

Para fazermos uma comparação de valores e tipos em JavaScript

usamos “===“.

Ex.:

como declarar: "0" === 0;

como ler: “0” tem o valor e o tipo idêntico a 0?

// nesse caso retorna false

**Operadores aritméticos**

São tipos de operadores matemáticos com valor numérico:

• + adição;

• - subtração;

• \* multiplicação;

• / divisão real;

• % divisão inteira;

• \*\* potenciação;

var adicao = 1 + 1;

console.log(adicao);

var subtracao = 2 - 1;

console.log(subtracao);

var multiplicacao = 2 \* 3;

console.log(adicao);

var divisaoReal = 6 / 2;

console.log(divisaoReal);

var divisaoInteira = 5 % 2;

console.log(divisaoInteira);

var potencicao = 2 \*\* 10;

console.log(potencicao);

**Operadores relacionais**

São tipos de operadores que consultam a relação entre valores:

• > maior que;

• < menor que;

• >= maior ou igual a;

• <= menor ou igual a;

var maiorque = 5 > 2;

console.log(maiorque);

var menorque = 5 < 2;

console.log(menorque);

var maiorOuIgual = 5 >= 2;

console.log(maiorOuIgual);

var menorOuIgual = 5 <= 5;

console.log(menorOuIgual);

**Operadores lógicos**

São tipos de operadores que consultam valores lógicos:

• && - “e” – considera que todos os valores sejam true;

• || - “ou” – considera que qualquer valor seja true;

• ! - “não” – inverte o valor de true para false ou vice-versa;

var e = true && false;

console.log(e);

var ou = true || false;

console.log(ou);

var nao = !true;

console.log(nao);

**Links**

Tipagem: <https://danvitoriano.medium.com/tipagem-dinâmica-no-javascript-e3551a445b38>

Variáveis:

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Grammar_and_types>

Escopo: <https://imasters.com.br/desenvolvimento/escopos-em-javascript>

Operadores: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/>

**Vetores e Objetos**

O que são Vetores ou Arrays

Arrays são um tipo de lista, ou matriz de variáveis, onde cada variável possui um índice. Os valores podem ser de vários tipos.

Imagine que um array é uma caixa com várias outras caixas dentro e cada uma contendo algum valor;

Ex: let array = ['string', 1, true]

console.log(array);

**Array**

O array deve ser declarado entre colchetes “[]”, e podem guardar qualquer valor dentro de seus índices: inclusive outros arrays.

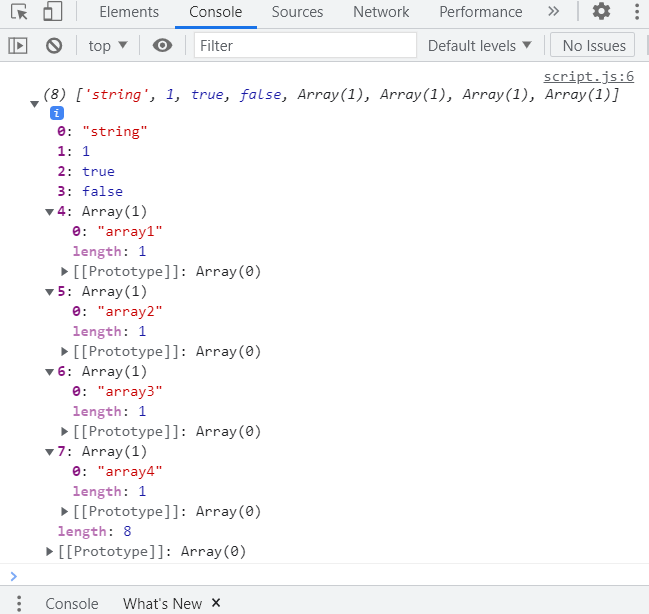
Ex

let array = ['string', 1, true, false, ['array1'], ['array2'], ['array3'],['array4']];

console.log(array);

Obs.: O índice só é acessado por um número inteiro, onde 0 é o primeiro índice. Cada índice é separado por vírgula.

console.log(array[3]); // ira pegar o índice 3



**Manipulando Arrays**

Ao ser declarado, o Array traz consigo uma série de métodos para manipulá-lo.

• forEach() – itera um array;

• push() – add item no final do array;

• pop() – remove item no final do array;

• shift() – remove item no início do array;

• unshift() – add item no início do array;

• indexOf() – retorna o índice de um valor;

• splice() – remove ou substitui um item pelo índice;

• slice() – retorna uma parte de um array existente;

let array = ['string', 1, true, false, ['array1'], ['array2'], ['array3'],['array4']];

console.log(array);

//forEach

array.forEach(function(item, index){

    console.log (item, index)

});

//push

array.push('novo item');

console.log(array)

//pop - apaga um item do final

array.pop();

console.log(array);

//shift - apaga um item do inicio

array.shift();

console.log(array);

//adiciona um item no inicio

array.unshift('novo item no inicio');

console.log(array)

//retorna o indice

console.log(array.indexOf(true));

//apaga alguns itens do array

array.splice(a, 3);

console.log(array);

let novoArray = array.slice(0,3);

console.log(novoArray);

**Objetos**

**O que são objetos?**

Objetos são dados que possuem propriedades e valores que definem suas características.

Deve ser declarado entre chaves “{}”.

Ex.: imagine uma xícara azul. Ela tem cor, pode ter vários tamanhos e funções.

Pode ser declarada assim:

var xicara = {

cor: ‘azul’,

tamanho: ‘p’,

funcao: tomarCafe()

}

**Manipulando objetos**

As propriedades de objetos podem ser atribuídas à variáveis, facilitando a manipulação do objeto. Chamamos isso de desestruturação.

Ex.: var xicara = {cor: ‘azul’, tamanho: ‘p’, funcao: tomarCafe()}

var cor = xicara.cor;

var tamanho = xicara.tamanho;

var funcao = tomarCafe();

Assim acessamos as propriedade do Objeto:

let object = {String: 'string', Number: 1, Boolean: true, array: ['array'], objectInterno:{

    objectInterno: 'objeto interno'}}

    console.log(object.Boolean);

var string = object.String;

console.log(string);

var arrayInterno = object.array;

console.log(arrayInterno);

Outra forma de fazer a desestruturação é utilizando chaves ao declarar a variável:

Ex.: var xicara = {cor: ‘azul’, tamanho: ‘p’, funcao: tomarCafe()}

var { cor, tamanho, funcao } = xicara;

let object = {String: 'string', Number: 1, Boolean: true, array: ['array'], objectInterno:{

    objectInterno: 'objeto interno'}}

    console.log();

var {string, boolean, objectInterno} = object;

console.log(string, boolean, objectInterno);

Para saber mais:

Arrays –

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array>

Objetos –

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Working_with_Objects>

Desestruturação –

<https://www.digitalocean.com/community/tutorials/understanding-destructuring-rest-parameters-and-spread-syntax-in-javascript-pt>

**Estruturas condicionais**

São instruções para realizar determinadas tarefas a partir de uma condição, seja de decisão ou repetição;

Ex.: Um jogo precisa mudar o placar toda vez que um jogador marca pontuação;

Se tal time marcou um gol, logo o placar precisa ser alterado.

**Estruturas de Decisão:**

**if**

Podemos usar as palavras reservadas “if” para estabelecer uma condição:

Ex.:

var jogador1 = 1;

var jogador2 = 0;

var placar;

if (jogador1 > 0) {

    console.log('Jogador 1 marcou ponto!');

}

// como ler: se o jogador1 tiver valor maior que 0 ele marcou ponto.

**else**

No caso de a condição não ser atendida podemos usar o “else”:

Ex.:

var jogador1 = 0;

var jogador2 = 0;

var placar;

if (jogador1 > 0) {

    console.log('Jogador 1 marcou ponto!');

} else{

    console.log('Ninguem marcou ponto');

}

**else if**

Caso haja mais de uma condição usamos o “else if“, para esta segunda condição

Ex.:

var jogador1 = 0;

var jogador2 = 1;

var placar;

if (jogador1 > 0) {

    console.log('jogador1 marcou ponto');

    } else if (jogador2 > 0) {

    console.log('jogador2 marcou ponto');

    } else {

    console.log('ninguém marcou ponto');

    }

**ninho de if**

Podemos também usar o “if” dentro de um outro “if”, chamamos isso de aninhamento de if’s ou ninho de if’s.

Ex.:

var jogador1 = 0;

var jogador2 = 0;

var placar;

if (jogador1 != -1) {

    if (jogador1 > 0) {

    console.log('jogador 1 marcou ponto');

    } else if(jogador2 > 0){

    console.log('jogador 2 marcou ponto');

    } else {

    console.log('ninguem marcou ponto');

    }

**If ternário**

Podemos também fazer uma verificação em uma única linha usando o “if” ternário:

Ex.:

var jogador1 = 1;

var jogador2 = 0;

var placar;

jogador1 != -1 && jogador2 != -1 ? console.log('Os jogadores são válidos'):

console.log('jogadores inválidos');

if (jogador1 > 0 && jogador2 == 0) {

    console.log('jogador 1 marcou ponto');

    placar = jogador1 > jogador2;

    } else if(jogador2 > 0 && jogador1 == 0){

    console.log('jogador 2 marcou ponto');

    placar = jogador2 > jogador1;

    } else {

    console.log('ninguem marcou ponto');

    }

// lembre de usar a interrogação “?” e dois pontos “:”

**Usando switch/case**

O “switch/case” funciona como uma estrutura condicional também;

Ex.: switch (${expressao}) {

case 1:

${instrucao};

break;

case 2:

${instrução};

break;

}

 switch (placar){

        case placar = jogador1 > jogador2:

        console.log('jogador 1 ganhou');

        break;

        case placar = jogador2 > jogador1:

        console.log('jogador 2 ganhou');

        break;

        default:

            console.log('Ninguem ganhou')

    }

**Laços de repetição**

São estruturas condicionais que repetem uma instrução até atingir determinada condição:

• For;

• For/in;

• For/of;

• While;

• Do/while;

**For**

Funciona como uma repetição de instrução até que a condição seja falsa:

for ([expressaoInicial]; [condicao]; [incremento]) { declaracao }

Ex.:

var array = [‘valor1’, ‘valor2’, ‘valor3’, ‘valor4’]

for (let i = 0; i < array.length; i++) {

console.log(i);

}

    let array = ['valor1', 'valor2', 'valor3', 'valor4', 'valor5',];

    let object = {propriedade1: 'valor1', propriedade2: 'valor2', propriedade3:

'valor3',}

//for executa uma instrução até que seja falsa

for (let indice = 0; indice < array.length; indice++){

    console.log(indice);

}

//imprime os indices

**For/in**

Funciona como uma repetição a partir de uma propriedade:

for ([indice] in [objeto ou array]) { declaracao }

Ex.:

var array = [‘valor1’, ‘valor2’, ‘valor3’, ‘valor4’]

for (i in array) {

console.log(i);

}

    let array = ['valor1', 'valor2', 'valor3', 'valor4', 'valor5',];

    let object = {propriedade1: 'valor1', propriedade2: 'valor2', propriedade3:

'valor3',}

//for/in executa repetição a partir de uma propriedade

//com array

for (let i in array){

    console.log(i);

}

//com object

for (i in object){ //percorre cada propriedade do objeto

    console.log(i);

}

**For/of**

Funciona como uma repetição a partir de um valor:

for ([indice] of [array]) { declaracao }

Ex.:

var array = [‘valor1’, ‘valor2’, ‘valor3’, ‘valor4’]

for (i of array) {

console.log(i);

}

//for/of - executa repetição a partir de valor

//com array

for (i of array) {

    console.log(i);

}

**For/of**

O For/of não funciona com objetos pois as propriedades variam, diferentes do índice em um array que sempre serão números inteiros.

Mas e se eu quiser pegar o valor mesmo assim?

Ex.:

for (i of object.propriedade) {

console.log(i);

}

// porém cada caractere dentro do valor será Impresso em linhas separadas.

//com objeto

for (i of object.propriedade1){

    console.log(i);

}

**While**

Executa uma instrução “enquanto” determinada condição for verdadeira, a verificação é feita antes da execução;

Ex.:

var a = 0;

while (a < 10) {

a++;

console.log(a);

}

// como ler: enquanto a variável a for menor que 10 ela vai receber mais um e imprimir no console.

var a = 0;

while (a < 10) {

a++;

console.log(a);

}

**Do/while**

Executa uma instrução “até que” determinada condição seja falsa, a verificação é feita depois da execução;

Ex.: var a = 0;

do {

a++;

console.log(a);

} while (a < 10)

// como ler: a variável a vai receber mais um e imprimir no console até que seu valor chegue a 10

**Links:**

Estruturas condicionais –

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/conditionals>

Laços de repetição –

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Loops_and_iteration>

**Funções e suas particularidades**

1. O que são funções?;

2. Como declarar funções;

3. Funções com parâmetros;

**Funções**

São blocos de comandos e instruções para a execução de determinadas tarefas:

Ex.: function nomeDaFuncao() {

${instrucao};

}

nomeDaFuncao();

**Como declarar?**

Geralmente se utiliza a palavra reservada “function” seguida de parênteses “()” e chaves “{}”:

Ex.: function funcao() {

console.log(“mensagem”);

}

funcao();

function funcao(){

    console.log('Tudo ok')

}

funcao();

() – indica que é um objeto do tipo function;

{} – indica que é um bloco de instrução;

**Funções com parâmetros**

As funções podem receber parâmetros em sua declaração, parâmetros, que servem como variáveis, onde sua atribuição pode ser feita durante a chamada da função:

Ex.: function nomeDaFuncao(parametro) {

${instrucao};

}

nomeDaFuncao(valorDoParametro);

Ex.: function mensagem(primeiro, segundo) {

console.log(primeiro, segundo);

}

nomeDaFuncao(“Tudo certo!”,”finalizamos”);

**Links:**

Funções –

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Functions>

**Tipos de Funções**

**Funções declarativas**

São funções que possuem o uso mais comum, deve ser declarada usando a palavra reservada “function” seguida do nome da função, parênteses “()” e chaves “{}”:

Ex.: function nomeDaFuncao() {

${instrucao};

}

// o nome da função é obrigatório;

//função declarativa que necessita de um nome

function funcao(){

    console.log('Mensagem de função declarativa')

}

funcao();

**Expressões de Funções**

São funções atribuídas à expressões. A nomeação das funções por expressão é opcional:

Ex. 1: var funcao = function nomeDaFuncao() {

${instrucao};

}

Ex. 2: var funcao = function() {

${instrucao};

};

//com nomeação

var funcao = function funcao(){

    console.log('Mensagem de função de expressão')

}

funcao();

se apagarmos o nome da função o js assume o nome da variável como o nome da função.

var funcao = function(){

**Arrow Function**

São funções de expressão de sintaxe curta.

Arrow functions sempre serão anônimas, e portanto não podem ser nomeadas.

Deve ser declarada com parênteses "()", seguido de "=>" e depois chaves "{}"

Ex. 1: var funcao = () => {

${instrucao};

}

var funcao =  () => {

    console.log('Sou uma arrow function')

}

funcao();

**Criando uma calculadora**

Para este exercício vamos utilizar alguns métodos nativos do JavaScript:

• Number() – para converter valores em números;

• Prompt() – para registrar entradas de usuário;

• Alert() – para mostrar mensagem ao usuário;

• Template Strings – para usar strings junto com expressões;

Calculadora.js

// criando uma calculadora

function calculadora() {

    // definindo as operações e pegando valor de entrada do usuário

    // usamos "\n" para fazer uma quebra de linha

    const operacao = Number(prompt('Escolha uma opção:\n 1 - soma (+)\n 2 - subtração (-)\n 3 - multiplicação (\*)\n 4 - divisão real (/)\n 5 - divisão inteira (%)\n 6 - potenciação (\*\*)'));

 // escolhendo operação

    // usand if && else

    if (operacao == 1) {

        soma();

    } else if (operacao == 2) {

        subtracao();

    } else if (operacao == 3) {

        multiplicacao();

    } else if (operacao == 4) {

        divisaoReal();

    } else if (operacao == 5) {

        divisaoInteira();

    } else if (operacao == 6) {

        potenciacao();

    }

 // definindo variáveis

        // todas as entradas devem ser do tipo number

        let n1 = Number(prompt('Insira o primeiro valor:'));

        let n2 = Number(prompt('Insira o segundo valor:'));

        let resultado;

 function soma() {

                resultado = n1 + n2;

                // mostrando o resultado usando template strings

                alert(`${n1} + ${n2} = ${resultado}`);

                }

calculadora();

Completa:

// criando uma calculadora

function calculadora() {

    // definindo as operações e pegando valor de entrada do usuário

    // usamos "\n" para fazer uma quebra de linha

    const operacao = Number(prompt('Escolha uma opção:\n 1 - soma (+)\n 2 - subtração (-)\n 3 - multiplicação (\*)\n 4 - divisão real (/)\n 5 - divisão inteira (%)\n 6 - potenciação (\*\*)'));

    // verificando se a operação é válida

    if (!operacao ||  operacao >= 7) {

        alert('erro - operação inválida');

        calculadora();

    } else {

        // definindo variáveis

        // todas as entradas devem ser do tipo number

        let n1 = Number(prompt('Insira o primeiro valor:'));

        let n2 = Number(prompt('Insira o segundo valor:'));

        let resultado;

        // verificando se as variáveis são válidas

        if (!n1 || !n2) {

            alert('erro - parâmetros inválidos');

            calculadora();

        } else {

            // definindo as funções

            function soma() {

                resultado = n1 + n2;

                // mostrando o resultado usando template strings

                alert(`${n1} + ${n2} = ${resultado}`);

                novaOperacao();

            }

            function subtracao() {

                resultado = n1 - n2;

                alert(`${n1} - ${n2} = ${resultado}`);

                novaOperacao();

            }

            function multiplicacao() {

                resultado = n1 \* n2;

                alert(`${n1} \* ${n2} = ${resultado}`);

                novaOperacao();

            }

            function divisaoReal() {

                resultado = n1 / n2;

                alert(`${n1} / ${n2} = ${resultado}`);

                novaOperacao();

            }

            function divisaoInteira() {

                resultado = n1 % n2;

                alert(`O resto da divisão entre ${n1} e ${n2} é igual a ${resultado}`);

                novaOperacao();

            }

            function potenciacao() {

                resultado = n1 \*\* n2;

                alert(`${n1} elevado à ${n2}ª potência é ${resultado}`);

                novaOperacao();

            }

            // nova operação

            function novaOperacao() {

                let opcao = prompt('Deseja fazer outra operação?\n 1 - Sim\n 2 - Não');

                if (opcao == 1) {

                    calculadora();

                } else if (opcao == 2) {

                    alert('Até mais!');

                } else {

                    alert('Digite uma opção válida!');

                    novaOperacao();

                }

            }

        }

    }

     // escolhendo operação

    // usand if && else

    if (operacao == 1) {

        soma();

    } else if (operacao == 2) {

        subtracao();

    } else if (operacao == 3) {

        multiplicacao();

    } else if (operacao == 4) {

        divisaoReal();

    } else if (operacao == 5) {

        divisaoInteira();

    } else if (operacao == 6) {

        potenciacao();

    }

/\*     // usando switch case

    switch (operacao) {

        case 1:

            soma();

            break;

        case 2:

            subtracao();

            break;

        case 3:

            multiplicacao();

            break;

        case 4:

            divisaoReal();

            break;

        case 5:

            divisaoInteira();

            break;

        case 6:

            potenciacao();

            break;

    }\*/

}

calculadora();

**Links:**

Funções –

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Functions>

Number –

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/NumberWindow.prompt>

<https://developer.mozilla.org/pt-BR/docs/Web/API/Window/prompt>

Window.alert – <https://developer.mozilla.org/pt-BR/docs/Web/API/Window/alert>

Template String -

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Template_literals>