**JavaScript**

**CONHECENDO O JAVASCRIPT**

Relação ***Client* x *Server***: Cliente seria o usuário do celular, notebook, aparelho etc. Servidor seria o fornecedor dos dados por trás das plataformas, aplicativos, programas etc. E o Javascript é uma tecnologia que funciona muito mais no lado do cliente.

Os websites são fundamentados em basicamente três tecnologias: **HTML** – responsável pelo conteúdo (texto, imagem, vídeo e suas respectivas estruturações e organizações). **CSS** – responsável pelo design do conteúdo (organizar cores, formatos, fontes, tamanhos, posições). **Javascript** – responsável pela engenharia da entrega, pela configuração da entrega.

Dentro de <***head***>(cabeça do site, onde são feitas configurações) tem um lugar onde alteramos o nome da barra do site, no campo <***title***>.

CONTEÚDO – Dentro do campo <***body***> (corpo) podemos colocar conteúdo (**HTML**): título (através do comando <***h1***>), parágrafo (através do comando <***p***>), etc.  
Ex.: <***h1***> Olá, mundo <***h1***>  
<***p***> Me livrei da Maldição. <***p***>

**\* os comentários em HTML são feitos com <!- - - ->**

ESTILO - Dentro do campo <***head***>, podemos colocar o estilo do conteúdo (**CSS**) através do comando <***style***>.  
Ex.: <***style***> body {  
 background-color: blue; **//coloca o fundo azul** color: White; **//coloca a letra branca** font: normal 20 pt Arial; **//coloca a fonte e altera o tamanho da letra** width: 200px; **//altera a largura** heigth: 200px **//altera a altura** text-align: 200px **//deixa no centro da tela** line-heigth: 200px **//deixa no meio da tela** }  
<***style***>

**\*os comandos em CSS deve finalizar com ;  
\* os comentários em CSS são feitos com /\* \*/**

INTERAÇÃO – No final do campo <***body***> através do comando <***script***>  
Ex.: <***script***>   
 window.alert(‘Minha primeira mensagem’) **//coloca um pop up com o alerta** window.confirme(‘Está gostando do conteúdo?’)  **//aparece um pop up de confirmação** window.prompt(‘Qual é o seu nome?’)   **//aparece um pop up p/ inserir mensagem**  
<***script***>

**COMANDOS BÁSICOS DO JAVASCRIPT**

**//** - é utilizado para fazer comentário em uma linha  
**/\* \*/** é utilizado para fazer comentário envolvendo mais de uma linha. Um é colocado no início e o outro no fim

**Variáveis** podem ser representadas por ***var*** ou por ***let***.

O JS não diferencia os tipos de números (inteiro – int, real – float). Para o JS todos os números são classificados como ***number***.

**Data types (tipos de dados**):  
number : infinity – NaN  
string  
boolean  
null  
undefined  
object: Array  
function

**MANIPULAÇÃO DE DADOS**

Em JS os comandos recebidos por um *prompt* são recebidos como *string*. Quando um número é recebido pelo prompt é necessário converter essa *string* em *number*. Isso é feito das seguintes formas:  
Number.parseInt(n) (ou só *parseInt(n)) //***para converter a string em número inteiro**Number.parseFloat(n) (ou só *parseFloat(n)) //***para converter a string em número flutuante**Number(n) *//***para deixar que para o próprio JS a conversão sem ser específica (inteiro ou flutuante)**Ex.:

var n1 = Number.parseInt(window.prompt('Digite o primeiro número: '))

var n2 = parseInt(window.prompt('DIgite o segundo número: '))

var n1 = Number.parseFloat(window.prompt('Digite o primeiro número: '))

var n2 = parseFloat(window.prompt('DIgite o segundo número: '))

var n1 = Number(window.prompt('Digite o primeiro número: '))

var n2 = Number(window.prompt('DIgite o segundo número: '))

**Conversão de *number* para *string*:**  
string(n) ou n.tostring()

**Concatenação** – Em JS pode ser feita da forma tradicional juntando item por item ou pode ser feita de forma mais otimizada através template string, utilizando Placeholder: ${ }  
Ex:  
Em vez de codar assim:  
> var nome = 'Alan'  
> var idade = 25  
> var nota = 9.8  
> 'O aluno ' + nome + ' que tem ' + idade + ' anos de idade tirou a nota ' + nota  
**'O aluno Alan que tem 25 anos de idade tirou a nota 9.8'**

Pode ser simplificado para:  
> `O aluno ${nome} que tem ${idade} anos de idade tirou nota ${nota}`  
**'O aluno Alan que tem 25 anos de idade tirou a nota 9.8'**

**Formatações para *strings*:**  
var s = ‘JavaScript’  
s.*length* //para contar quantos caracteres tem na string  
s.*toCase()* //para deixar tudo em maiúsculo  
s.*toLowerCase()* //para deixar tudo em minúsculo

Ex. de formatação de strings:

var nome = window.prompt('Qual é o seu nome? ')

document.write(`Olá, <strong>${nome}</strong>! O seu nome tem ${nome.length} letras.<br/>`)

document.write(`Seu nome em maiúsculo fica: ${nome.toUpperCase()} e em minúsculo fica: ${nome.toLowerCase()}`)

A tag <Strong> </Strong> faz com que o item entre ela fique em negrito  
A tag <br/> faz com que pule a linha

**Formatações para *Numbers*:**var n = ‘1545.5’  
n.toFixed(2) //para colocar duas casas decimais no número  
n.toFixed(2).replace(‘.’ , ‘,’) //para trocar um caractere por outro. Nesse caso o ponto pela virgula  
n.toLocaleString(‘pt-br’, {style: ‘currency’, currency: ‘BRL’}) //para converter para real  
n.toLocaleString(‘pt-br’, {style: ‘currency’, currency: ‘USD}) //para converter para dolar  
n.toLocaleString(‘pt-br’, {style: ‘currency’, currency: ‘EUR}) //para converter para euro

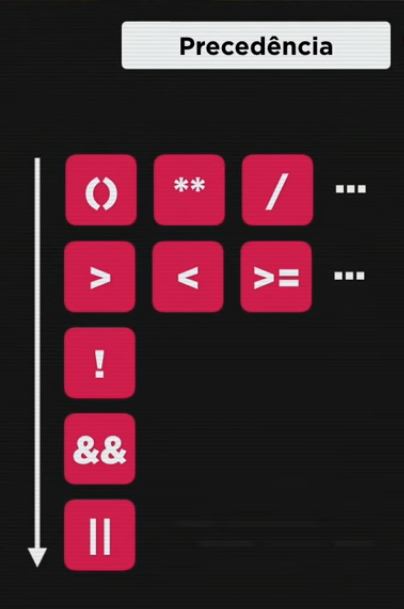
**OPERADORES**

**Aritméticos**  
+ Soma  
- Subtração  
\* Multiplicação  
/ Divisão  
% Resto da divisão  
\*\* Potencialização

Ordem de precedência:  
1° ( )  
2° \*\*  
3° \* / %  
4° + -

**Relacionais –** Toda expressão que tem um valor relacional, o resultado será sempre um valor booleano (verdadeiro ou falso)  
> Maior  
< Menor  
>= Maior ou igual  
<= Menor ou igual  
= = Igual mesmo sendo um number e uma string Ex.: 5 == ‘5’ ***true***  
!= Diferente  
= = = igual idêntico tanto valor quanto a tipagem do mesmo Ex.: 5=== ‘5’ ***false***!= = desigual restrito onde os valores são de tipos diferentes, mas o valor é o mesmo

**Lógicos**  
! Negação resultado: **false** ou **true**  
&& Conjunção (e) resultado: só uma opção será **true**  
|| Disjunção (ou) resultado: só uma opção será **false**



**Ternário –** Tem três partes: bloco de teste lógico que dará verdadeiro ou falso, bloco do meio que é o que vai acontecer por consequência do resultado do primeiro bloco e o terceiro bloco que diz o que vai acontecer caso o teste lógico seja outro resultado.  
?  
:  
Ex1:   
var média = 5.5  
média>= 7 **?** “Aprovado” **:** “Reprovado”  
***Reprovado***  
Ex.2:  
var x = 8  
var resp = x %2 == 0? 5 : 9  
***5***Ex.3:  
var idade = 19  
var r = idade >= 18 ? ‘MAIOR DE IDADE’ : ‘MENOR DE IDADE’  
***MAIOR DE IDADE***

**DOM – *Document Object Model* (vertente web do JavaScript)**

Dom é um conjunto de objetos dentro do navegador que vai dar acesso aos componentes internos do website.

**Árvore DOM:**Raiz da árvore – ***window*** (tudo do JavaScript está dentro da window)

Dentro de ***window*** tem vários objetos. Ex.: ***location*** que diz onde está localizado o site – URL, página atual, página anterior, ***document*** que é o documento atual, ***history*** que é onde guarda de onde veio, para onde vai etc.

Dentro de ***document*** está o HTML

Dentro de ***HTML*** há o ***head*** e o ***body.*** O HTML é o parent (pai) do head e do body. Já o head e o body são child (crianças, filhos)

Dentro de ***head*** há o ***meta*** e o ***title***

Dentro do ***body*** pode haver o ***h1, p, div***

Dentro do ***p*** pode haver uma ***Strong***

***EXEMPLO de uma Árvore DOM:  
Diagrama

Descrição gerada automaticamente***

Os elementos DOM podem ser selecionados por:  
Marca ID Nome Classe Seletor

Seleção por **Marca** (ou por tag name) – Comando *getElementsByTagName( )*. Consegue selecionar mais de um objeto através da identificação pelos colchetes [ ].

Ex.:

        var p1 = document.getElementsByTagName('p')[0]

        document.write('Está escrito assim: ' + p1.innerText)

O comando document.getElementsByTagName(‘p’) vai selecionar o parágrafo e o [0] está indicando que o parágrafo selecionado é o primeiro. Caso fosse [1] o parágrafo selecionado seria o segundo.  
O comando document.write(‘Está escrito assim: ’ + p1.inerText) está mostrando na tela o parágrafo selecionado, que no caso foi o primeiro parágrafo.  
innerText pega o texto que está dentro do elemento que selecionamos, porém sem alguma formatação que tenha nele. Para pegar com formatação se usa o ***innerHTML.***  
Ex. document.write(p1.innerHTML)

Em vez de escrever na tela o elemento selecionado, podemos fazer alteração nesse elemento que foi selecionado.  
Ex.1:

var p1 = document.getElementsByTagName('p')[0]

        p1.style.color = 'blue'

Ex.2:

        var corpo = document.body

        var p1 = document.getElementsByTagName('p')[0]

        corpo.style.background = 'black'

        p1.style.color = 'blue'

Seleção por **ID** – Comando *getElementById( )*. É necessário que haja identificação do elemento para poder selecioná-lo via Id. Só seleciona um elemento.

Seleção por **Name** – Comando *getElementsByName( )*. É necessário que haja uma propriedade name no elemento para poder selecioná-lo dessa forma. Selecionar mais de um elemento.

Seleção por **Classe** – Comando *getElementsByClassName( )*. Seleciona mais de um elemento.

Seleção por **Seletor** – Comando *querySelector( )* ou *querySelectorAll( )*Toda id é representada por uma hashtag. Ex.: (‘div#msg’)  
Toda class é representada por um ponto. Ex.: (‘div.msg’)

**Como colocar uma identificação em um elemento?**  
A forma de identificar os elementos é a mesma para os diferentes tipos de elementos: ID, Name, Marca, Classe. Para identificar é só colocar dentro do elemento a identificação.  
Ex.: <div> Clique aqui </div> **//elemento natural**  
<div id = “msg” > Clique aqui </div> **//elemento identificado por “msg”**

**EVENTOS DOM**

Evento DOM é tudo aquilo que pode acontecer com o elemento. Ele pode ser colocado tanto na parte de HTML quanto na parte de JavaScript. Para colocar o evento detro do HTML, deve colocar o evento dento do elemento (Ex.: <div id= “área” ***oneclick = “clicar( )”***>)  
Ex.: Evento de mouse:  
mouseenter – onde o mouse entrou no elemento  
mousemove – quando o mouse se move dentro do elemento  
mousedown – quando clica e segura o mouse dentro do elemento  
mouseup – quando solta o clique  
click – quando clica  
mouseout – quando clica fora do elemento

\***pesquisar por JavaScript DOM elevents list (event reference, da Mozila)– para pesquisar as possibilidades de eventos.**

**FUNÇÃO**

Conjunto de códigos que vão ser executadas quando o evento ocorrer.

No contexto de função, se houver um bloco de 10 linhas de código, esse bloco funcionará estando dentro de chaves { } quando ocorrer o evento determinado. Esse bloco deve ser nomeado como *function* tendo como complemento a ação da função.  
Ex.: function ação ( ) {  
 bloco de linhas }

Explicação dos comandos de **função** abaixo **colocada no HTML**:

 <div id="area" onclick="clicar()" onmouseenter="entrar ()" onmouseout="sair()">

        Interajaaaaa

    </div>

    <script>

        function clicar() {

            var area = document.getElementById('area')

            area.innerText = 'Clicou!'

            area.style.background = 'red'

        }

        function entrar () {

            var area = document.getElementById('area')

            area.innerText = 'Entrou'

        }

        function sair() {

            var area = document.getElementById('area')

            area.innerText = 'Saiu!'

            area.style.background = 'blue'

A div que está dentro do HTML foi identificada como “área” e dentro dela foram colocadas três funções: onclick (nomeada como clicar) para quando houve um clique na div, a função onmouseenter (nomeada como entrar) para quando o mouse entrar no espaço da div e a função onmouseout (nomeada como sair) para quando o mouse sair de dentro da div.

Dentro do JS (script) as funções colocadas no HTLM foram desenvolvidas e direcionadas.  
Função clicar ( ) – essa função sendo acionada, foi criado uma variável chamada área fazendo uma ligação com a div de id “area” e essa variável foi transformada em clicou e seu background transformado em vermelho.  
Função entrar ( ) – essa função sendo acionada, foi criado uma variável chamada área fazendo uma ligação com a div de id “area” e essa variável foi transformada em entrou e seu.  
Função sair ( ) – essa função sendo acionada, foi criado uma variável chamada área fazendo uma ligação com a div de id “area” e essa variável foi transformada em sair e seu background transformado em azul.

Explicação dos comandos de **função** abaixo **colocada no Script**:

    <div id="area">

        Interajaaaaa

    </div>

    <script>

        var area = document.getElementById('area')

        area.addEventListener('click', clicar)

        area.addEventListener('mouseenter', entrar)

        area.addEventListener('mouseout', sair)

        function clicar() {

            var area = document.getElementById('area')

            area.innerText = 'Clicou!'

            area.style.background = 'red'

        }

        function entrar () {

            var area = document.getElementById('area')

            area.innerText = 'Entrou'

        }

        function sair() {

            var area = document.getElementById('area')

            area.innerText = 'Saiu!'

            area.style.background = 'blue'

Há uma div que foi identificada como “área” e ela foi ligada a variável área que está dentro do script. Os eventos que darão a dinâmica da div foram criados dentro do script, tendo como comando ***addEventListener*** que foi colocado com os eventos *click, mouseenter* e *mouseout*, ligado as funções criadas *clicar, entrar* e *sair* .

Outro exemplo do uso da função:

<head>

    <title>Somando números</title>

    <style>

        body {

            font: normal 16pt Arial;

        }

        input {

            font: normal 16pt Arial;

            width: 100px;

        }

        div#res {

            margin-top: 20px;

        }

    </style>

</head>

<body>

    <h1>Somando os Valores</h1>

    <input type="number" id="txtn1"> +

    <input type="number" id="txtn2">

    <input type="button" value="Somar" onclick="somar()">

    <div id="res">Resultado:</div>

    <script>

        function somar() {

            var tn1 = document.getElementById('txtn1')

            var tn2 = document.getElementById('txtn2')

            var res = document.getElementById('res')

            var n1 = Number(tn1.value)

            var n2 = Number(tn2.value)

            var s = n1 + n2

            res.innerHTML = `A soma entre ${n1} e ${n2} é igual a <strong>${s}</strong>.`

**HTML**: foi criado um título chamado “Somando os valores”. Foram criadas duas caixas de entrada de dados e um botão com texto “somar” e com a função de click com nome “Somar”.

Foi criada uma div chamada de “resultado”. Essa div foi identificada como “res”

**JAVASCRIPT**: Foi colocada a função somar e dentro dela foram criadas duas variáveis para receber os valores colocados na caixa de entrada, uma variável para guardar o valor da div “res”. Foram criadas duas variáveis para transformar os valores das caixas de entrada em número, pois os valores das caixas de entrada vêm como string. Foi criada uma variável para somar os valores das caixas de entrada convertidos em número e no fim a div “res” foi transformada no texto de exibição dos valores da soma.

**CSS**: A fonte do body foi formatada. A fonte dos inputs foi formatada e o tamanho horizontal da tag foi ajustado. A div teve o topo da sua margem ajustado.

**Como Detectar um erro no JS?**

Na maioria dos casos clica com o botão direito no site -> clica em inspect -> vai aparecer na tela DevTools um x informando um erro e em baixo no console vai mostrar o erro e em que linha ele está.

**CONDIÇÕES**

(Para abrir o nodeJS direto no Visual Studio Code, basta instalar o plugin NodeJs Exec e no novo arquivo criado como js apertar F8 para executar os comandos.)

Forma de pegar a hora do sistema:  
Cria uma variável recebendo o comando **new Date**();  
Cria outra variável, esse recebendo o comando a variável acima e o comando **getHours**()  
Ex.: var agora = New Date()  
var hora = agora.getHours()

Há um comando no JS chamado ***switch*** que serve como o if, onde há dentro de seu bloco possibilidades que serão executadas caso determinado valor vá de acordo com a expressão.  
O switch é importante para fazer condições pontuais que usando o if faria com que o código fosse maior e mais trabalhoso.  
O switch só funciona com números inteiros e caracteres.  
Ex.: switch(expressão) {  
 case valor 1:  
  
 break  
  
 case valor 2:  
  
 break  
  
 case valor 3:  
  
 break  
 default:

Se a expressão corresponder ao valor 1, será executado o bloco do case valor 1. Se a expressão corresponder ao valor 2, será executado o bloco 2 e assim por diante. default corresponde ao else.

**\*Obs.: Dentro de cada bloco do comando switch tem que ter um comando break. Pois é esse comando que fará com que o código continue sendo executado depois de executado o comando switch. No bloco default não é obrigatório colocar o comando break, dos demais sim.**

Exemplo do uso do ***switch***

var agora = new Date()

var diasem = agora.getDay()

switch (diasem) {

    case 0:

        console.log('Domingo')

        break

    case 1:

        console.log('Segunda')

        break

    case 2:

        console.log('Terça')

        break

    case 3:

        console.log('Quarta')

        break

    case 4:

        console.log('Quinta')

        break

    case 5:

        console.log('Sexta')

        break

    case 6:

        console.log('Sábado')

        break

    default:

        console.log('ERRO. Dia inválido')

}

Foram criadas duas variáveis: Uma para receber o comando da data e a outra para pegar o dia do comando data da primeira variável. Foi criado o comando switch que representava a variável do dia e nele foram colocados os blocos para representar o dia da semana.  
O comando **getDay()** representa os dias da semana por números, sendo o primeiro dia da semana representado por 0: 0 – Domingo; 1 – Segunda... 6 – Sábado.  
Os cases foram colocados como números para serem executados de acordo com o número do respectivo dia.

**REPETIÇÃO**

A estrutura mais básica do um laço de repetição é composto por um teste lógico e em seguida um bloco de comandos (**while**).  
Ex.: var c = 1 //contador começando em 1  
while (c >= 3) { //enquanto o contador for menor ou igual a 3  
 console.log(`Passo $(c).`) //mostra na tela uma mensagem  
 c++ //e acrescenta mais 1 ao contador  
}  
**Passo 1.  
Passo 2.  
Passo 3.**

Há outra estrutura de repetição que é o contrário da mais básica. Nessa, em vez de primeiro fazer o teste lógico, ela primeiro executa o bloco e depois do bloco executado ela faz o teste lógico (**do...while**).  
Ex.: var c = 1  
do { //do quer dizer faça  
 console.log(`Passo $(c).`) //mostra na tela uma mensagem  
 c++ //e acrescenta mais 1 ao contador  
} while (c >= 3) //enquanto o contador for menor ou igual a 3

Há também outra estrutura de repetição, essa com variável de controle (**for**). Onde são feitos uma inicialização, um teste lógico e um incremento. Na primeira passagem é feita a inicialização e o teste lógico. Caso o teste lógico seja verdadeiro, a passagem irá voltar para a inicialização e irá ser feito um incremento.  
Ex.: for (var c = 1; c <= 5; c++) {  
 console.log(`passo$()`)  
}

O vscode permite fazer a **depuração** dos códigos, que é uma forma de ver passo a passo os códigos sendo execultados. Para isso deve clicar em start debugging (ou F5), clicar no besourinho que aparece no canto esquerdo da tela, tirar o terminal e colocar no console do debug (debug console), marcar com a bolinha vermelha a linha que a partir dela será acompanhada, no campo da esquerda chamado watch, deve colocar a variável que quer monitorar, e para que ela possa ser monitorada deve rodar o programa apertando em F10.

**VARIÁVEIS COMPOSTAS**

Também chamada de *arrays* (vetor), armazena mais de um valor sem perder o valor anterior. A forma de expressar uma array é através dos colchetes, onde cada valor (string ou number) dentro deles é representado por um índice (chave) que começa no 0.  
Ex.1: var auto = [azul, amarelo, rosa, branco, preto]  
O primeiro valor dentro da array é representado por 0, o segundo por 1, e assim por diante.

Interface gráfica do usuário, Diagrama

Descrição gerada automaticamente

Ex.2: let num = [2, 5, 8, 6]  
console.log(`Nosso vetor é ${num}.`  
**Nosso vetor é 2, 5, 8, 6.**

É possível adicionar novos valores nos arrays através de atribuição desse valor à variável da array junto com a identificação do índice (chave) onde esse novo valor será acrescentado.  
Ex.: let num = [2, 5, 8, 6]  
let num[4] = 8 **//aqui é identificado que o valor 8 é recebido na chave 4, ou   
 seja, no espaço vazio da array, fazendo com que esse número  
 seja acrescentado.**  
console.log(`Nosso vetor é ${num}.`  
**Nosso vetor é 2, 5, 8, 6, 8.**

Uma outra forma de adicionar algum valor a uma array é através do comando ***push*** que faz o acréscimo independente de onde seja o espaço vazio na array.  
Ex.: let num = [2, 5, 8, 6]  
num.push(7)  
console.log(`Nosso vetor é ${num}.`  
**Nosso vetor é 2, 5, 8, 6, 7.**

Uma forma de descobrir o cumprimento da array é através do atributo ***length()***.  
Ex.: let num = [2, 5, 8, 6]  
console.log(`Nosso vetor tem ${**num.length**} posições.`  
**Nosso vetor tem 4 posições.**

Uma forma de colocar os valores da array de forma crescente é através do atributo ***sort().***Ex.: let num = [2, 5, 8, 6]  
num.sort()  
console.log(num)  
**2, 5, 6, 8**

Uma forma simplificada de mostrar cada valor que está dentro da array é através do for.

let num = [8, 1, 9, 7, 2, 6]

for (let pos = 0; pos < num.length; pos++) {

    console.log(`A posição ${pos} tem o valor ${num[pos]}.`)

}

Foi criada uma array. Foi criado um for e nele uma variável para pegar a posição de cada valor sendo igual a zero (pois o índice começa em 0). Essa variável foi colocada um teste lógico dela ser menor que o cumprimento da array e acrescentado mais 1 para ela seguir para a próxima posição da array.  
No console.log foi colocado para mostrar a posição que estava sendo passada pelo for, junto com o valor da array guiado pela sua respectiva posição.

Há uma forma de simplificar ainda mais a forma de mostrar na tela o que está dentro da array. Também com o for, porém com o ***in*** na estrutura dele, usando somente dois elementos em vez do tradicional que são três elementos:

let num = [8, 1, 9, 7, 2, 6]

for (let pos in num) {  //"para cada posição dentro da array num"

    console.log(`A posição ${pos} tem o valor ${num[pos]}.`)

}

Dentro do for in, para cara posição dentro da variável num (em num) o console.log mostra na dela a posição do valor com o respectivo valor.

Há uma forma de saber se dentro da array tem determinado valor, mostrando em que posição esse valor se encontra dentro da array. Isso é feito através da atribuição ***indexOf().***  
Ex.: let num = [2, 5, 8, 6]  
let pos = num.indexOf(8)  
console.log(`O valor 8 está na posição ${pos}.`  
**O valor 8 está na posição 2.**

Quando o valor não é encontrado, o JS dá como resposta -1. Nesse caso pode ser feito um macete para mostrar na tela que o valor não foi encontrado, utilizando if... else.

let num = [5, 9, 8, 1, 2]

let pos = num.indexOf(5)

if (pos == -1) {

    console.log('Valor não encontrado.')

} else {

    console.log(`O valor está na posição ${pos}.`)

}

**OBJETOS**

Os objetos servem para alcançar um limite que as arrays não alcançam.

O índice das arrays são sempre limitados ao número das “casas”. Já os objetos permitem declarar o nome de cada casa. Isso é feito através de chaves em vez de colchetes.  
Ex.: let amigo = {nome: ‘José’, Sexo: ‘M’, Peso: ‘85.2’, engordar(p){}}  
Isso cria um espaço na memória da variável chamado nome com o elemento José, cria um espaço chamado sexo com o elemento M, cria um espaço chamado peso com o elemento 85.2 e ainda através dos objetos é possível criar uma função dentro dele, nesse caso uma função chamada engordar com parâmetro p.

**FUNÇÕES**

Toda função pode ter: uma chamada que pode ser “vocal” ou automática, o parâmetro (que pode ser mais de um) que faz parte da chamada, a ação e o retorno. Em outras palavras, funções são ações executadas assim que são chamadas ou em decorrência de algum evento.

A chamada pode estar fora de função e a partir dela a função é executada.

function parimp(n) {

    if (n % 2 == 0) {

        return 'par'

    } else {

        return 'impar'

    }

}

let res = parimp(11)

console.log(res)

Foi criada uma função chamada parimp tendo como parâmetro n. para que essa função funcione será necessário a criação a chamada para ela, nesse caso foi criada fora dela através da variável res colocando o parâmetro da function parimp como 11.  
Dentro da função foram colocadas as condições para saber se o número é par ou ímpar. Como na chamada foi colocado um número ímpar (11) o retorno será ímpar, caso na chamada fosse colocado um número par, o retorno seria par.  
Por fim foi colocado para mostrar na tela o valor da resposta. Se o retorno do número for que ele é par, a variável res passará a valer “par”, se o retorno do número for que ele é ímpar, a variável res passará a valer “ímpar”. Para mostrar na tela poderia ser feito direto sem criar uma variável para resposta. Seria: ***console.log(parimp(11))***, dessa forma já iria ser condicionado o número do parênteses e mostrado em que ele se encaixa (par ou ímpar).

/\*function soma(n1, n2) {

    return n1 + n2

}

console.log(soma(2, 5))

Essa forma de fazer pode trazer um problema, pois

se alguns dos parâmetros não foram colocados, o

resultado será NaN\*/

function soma(n1 = 0, n2 = 0) {

    return n1 + n2

}

console.log(soma(2))

/\*Dessa forma mesmo que um dos parâmetros não seja colocado, a "vaga"

dele será preenchida com 0 \*/

Uma função pode estar dentro de uma variável:

let v = function(x) {

    return x\*2

}

console.log(v(5))

Outras formas de utilizar a function:

Calcular fatorial de um número de forma simples:

function fatorial(n) {

    let fat = 1

    for (let c = n; c > 1; c--) {

        fat \*= c

    }

    return fat

}

console.log(fatorial(5))

Foi criada uma function chamada fatorial recebendo o parâmetro n.  
Dentro dela foi criada uma variável fat sendo igual a 1.  
Foi criada também um for, e dentro dele uma variável contadora c sendo igual ao parâmetro, pois será a partir da chamada desse parâmetro que o fatorial começará a ser feito. O teste lógico do for foi que o contador que foi transformado no parâmetro fosse maior que 1, e o incremente foi que o contador passasse a valer 1 a menos a cada vez que passasse pelo laço.  
O fatorial foi calculado em seguida através da multiplicação de fat pelo número do parâmetro que se transformou em c.  
por fim foi dado o retorno de fat e mostrado na tela o fatorial.

É possível calcular o fatorial através de uma function chamar ela mesma (**Recursividade**):

function fatorial(n) {

    if (n == 1) {

        return 1

    } else {

        return n \* fatorial(n-1)

    }

}

console.log(fatorial(5))

Foi criada uma função chamada fatorial. Nela foi verificada se o parâmetro seria igual a 1, pois se for o retorno será 1. Caso não seja, o retorno será uma outra forma de calcular o fatorial: o parâmetro o multiplicará – 1 e depois mostrará na tela.

Estuda mais...

Interface gráfica do usuário

Descrição gerada automaticamente

------------------------------------------------------------------------------------------------------------------------------

**OUTROS/MAIS**

**HTML**

-Outra forma de adicionar imagens no HTML é:  
Criando a tag img; dentro dela acrescentando o src para utilizar uma fonte de informação ou arquivo; utilizando a barra( / ) para localizar a pasta da imagem e novamente a barra dessa vez para localizar a imagem.  
Ex.: <img scrc= “/imagens/as\_you\_were.webp”> </img>

-Uma div pode ser colocada dentro de outra div, criando uma subdivisão da div principal.  
Ex.:  
<div>  
 <div> Nome da música </div>  
 <div> Nome da banda </div>  
</div>

**CSS**

-É possível transformar um background num gradiente. Isso é feito através da criação de um background de imagem em vez de cor, transformando a cor numa imagem. Além de criar uma atribuição para background imagem, dentro dela deve ser colocado tanto a cor inicial do gradiente quanto a cor final. Por fim, para determinar o tamanho da altura vertical disponível, é usado a atribuição heigth, para que a imagem criada como background não fique repetindo o seu gradiente.  
Ex.: body {  
 background-image: linear-gradient (  
 rgb(59,88,93),  
 rgb(29,40,42)  
 );  
 heigth: 100vh;  
 }

Uma forma de alterar a fonte através do CSS é pela atribuição da família da fonte: “font-family:”

Uma forma de fazer com que o body se comporte em relação aos elementos dele é através da atribuição “display: flex;”. Isso faz com que todos os elementos do body mudassem de disposição, sendo colocados um ao lado do outro, em linha, de acordo com a ordem. Para alterar a direção do display flex é só direcionar o flex: “flex-direction: column”. Isso faz com que o site seja responsivo.  
Ex.:  
display: flex;  
flex-direction: column;

**BIBLIOTECA DE ICONES**

Pode ser utilizada a bootstrap icons. Ele mostrará uma série de ícones que podem ser utilizados no site. Ao fim dele, no campo CDN, tem um código para copiar e colar no head do HTML. Esse link “dirá” para o HTML “ficar de olho” na biblioteca que nele consta.

Ao escolher o ícone, abrindo-o, terá um código em Icon font. Esse código deverá ser copiado e colado no elemento desejado (parágrafo, div, etc.)