**Javascript**

Javascript é uma linguagem de programação que funciona do lado do cliente (navegador). Sendo responsável pela **interação** do usuário com a página, por exemplo, movimentar o mapa do google e ter as coordenadas alteradas na url ou um menu contraído que ao clicar dispara esse **evento** para o expandir o menu. Sendo importante mencionar que, o Javascript é uma linguagem **sem tipo fixo**, ou seja, aceita qualquer valor seja 19, 19.50, “texto”, ‘a’, Objeto{}.

A importação do arquivo Javascript é feito dentro do <head> pela tag: <script language=”javascript” src=”caminho de onde o arquivo está armazenado”></script>

// comentário de linha

/\*

comentário

de

múltiplas

linhas

\*/

Os comentários são importantes **durante** **o desenvolvimento** para descrever linhas de código, já que eles não executados como código.

var mensagem = "Olá, Mundo!";

alert(mensagem);

As variáveis são espaços ou endereços de memória associados a um nome para armazenar um determinado tipo de informação durante a execução do algoritmo ou programa. No caso do Javascript, elas são identificadas pelas palavras reservadas var, let, const e armazenam a informação após o sinal de ‘=’. Palavras reservadas :

* Var é uma variável mutável e com escopo global / local, ou seja, que recebe novos valores e acessível no arquivo inteiro ou somente onde foi criada, por exemplo, dentro de funções.
* Let é uma variável mutável e com escopo local, ou seja, que recebe novos valores e acessível somente onde foi criada, por exemplo, dentro de funções.
* Const é uma variável **i**mutável e com escopo local, ou seja, que **não** permite alteração e acessível somente onde foi criada, por exemplo, dentro de funções.

\*alert(“mensagem qualquer”); -> dispara uma janela com mensagem de aviso.

var a = 2;

var b = 3;

var soma =  a + b;

var sub =  a - b;

var multi = "Resultado da multiplicação de A ( 2 ) e B ( 3 ) é "+ a \* b;

var div = "Resultado da divisão de A ( 2 ) e B ( 3 ) é "+ a / b;

var resto = "Resultado do resto da divisão de A ( 2 ) e B ( 3 ) é "+ a % b;

alert("Resultado da soma de A ( 2 ) e B ( 3 ) é " + soma);

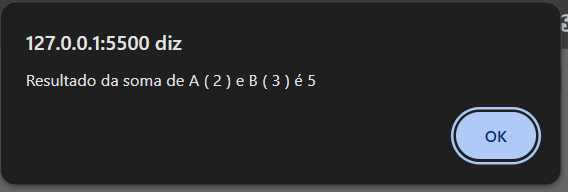
alert("Resultado da subtração de A ( 2 ) e B ( 3 ) é "+ sub);

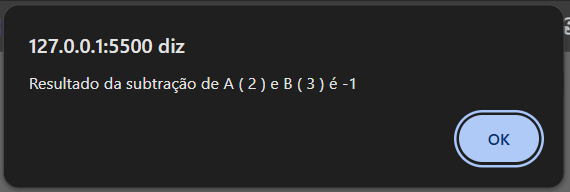
alert(multi);

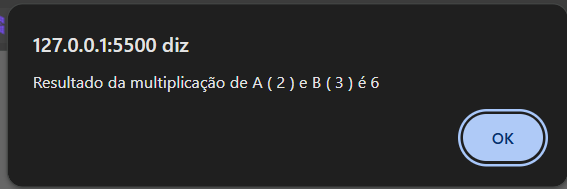
alert(div);

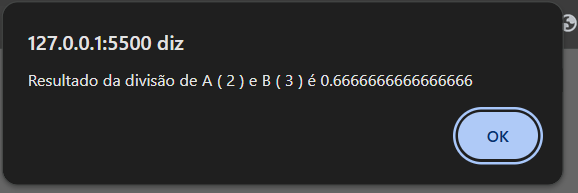
alert(resto);

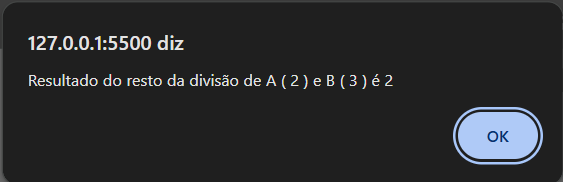
As operações matemáticas de mais, menos, multiplicação, divisão e módulo (resto da divisão) são representados pelos símbolos de +, -, \*, / e %. Podendo usar variáveis para armazenar estes valores, o resultado da operação e um texto explicativo concatenando (+) com o resultado.











A estrutura condicional aplica um(a) condição/ critério para executar uma ação em caso positivo (true) ou executar uma outra ação em caso negativo (false). Supondo a seguinte regra, caso(se) o módulo do número X por 2 seja (for) igual a 1, então o número é ímpar, se não, se o módulo do número X por 2 seja (for) igual a 0, então o número é par, se não o valor é inválido. Traduzindo em código, usa-se if ( variável % 2 == 1) { alert(“o número ”+variável+” é impar”) } else if ( variável % 2 == 0) { alert(“o número ”+variável+” é par”)} else alert(“o valor é inválido”); .

var a = 2;

var b = 3;

var c = "Olá, Mundo"

if ( b % 2 == 1 ){

    alert("O número "+ b +" é impar");

} else if (c % 2 == 0){

    alert("O número "+ c +" é par");

} else {

alert("Valor inválido");

}

Laços ou estrutura de repetição é responsável por repetir um trecho de código por um determinado número de vezes.

* While (condição){.... contador = contador + 1}: Enquanto a condição for verdadeira, então executa um trecho de código e no final incrementa o contador. Quando a condição for falsa, encerra o algoritmo.
* For ( var j = 0 ; condição; j++){ ...}: Para j valendo 0, enquanto a condição for verdadeira, executa um trecho de código, incrementa o j em mais 1 e verifica a condição novamente até que ela seja falsa, para encerrar o algoritmo.

var i = 0;

while (i < 3){

    alert(i);

    i = i + 1;

}

for ( var j = 0; j < 3; j++){

    alert(j);

}

Para tipos de dados, o tipo é definido **dinamicamente** pela atribuição:

* Número inteiro
* Número flutuante
* Texto
* Caractere
* Objeto
* Lista

var inteiro = 2;

var flutuante = 2.60;

var texto = "Olá, Mundo";

var caractere = "o";

var carro = {

    "modelo": "gm",

    "ano": 2009

}

var lista = ["laranja","maça","pera"];

alert(lista);

lista = [inteiro,flutuante,texto, caractere,carro.modelo,carro.ano];

alert(lista);

As listas, arranjos, ou Arrays são formas de agrupar uma sequência de elementos acessíveis por um ponteiro ou posição. Supondo que temos uma lista de frutas como laranja, maça, pera. Para acessar cada elemento desta lista, é preciso indicar a posição do item dentro da lista, lembrando que em geral, as posições começam por 0. Podendo usar o laço For para pegar estas posições, varrendo a lista dinamicamente, lendo da seguinte forma: para ( cada posição dentro da lista) então exibir o valor do item da lista baseado na posição . Desta forma o código seria assim:

var lista2 = ["laranja","maça","pera"];

alert(lista2[0]);

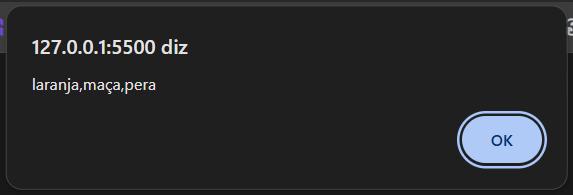
alert(lista2[1]);

alert(lista2[2]);

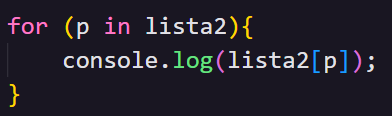
for (p in lista2){

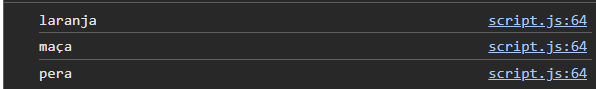
    alert(lista2[p]);

}



Por último, ao onvés de usar as janelas de aviso, podemos usar o log do console do navegador, por meio do seguinte código: console.log(“texto”/variável).





**Exercício extra: criar uma página para fazer cálculos matemáticos ( +, -, /, \*, %resto da divisão), simulando uma calculadora.**

**JQuery**

JQuery é uma biblioteca Javascript que irá conter várias funcionalidades prontas para uso, funcionando como framework, permitindo o desenvolvedor escrever menos linhas de código e fazer mais (programação funcional).

Para importar a biblioteca JQuery, usamos a tag script dentro do footer : <footer><script src="./src/jquery.min.js"></script></footer>.

O primeiro passo para utilizar as funções do JQuery é criar um novo arquivo script: script\_jquery.js. Importar este mesmo arquivo da mesma do Jquery.

Supondo que temos um título e um botão numa página html:

<!DOCTYPE *html*>

<html>

    <head>

        <meta *charset*="utf-8">

        <meta *name*="viewport" *content*="width=device-width,  initial-scale=1">

        <title>Usando a blibioteca jquery</title>

    </head>

    <body>

        <h1> Olá, Mundo!</h1>

        <button>Esconder Olá, Mundo!</button>

        <footer>

            <script *src*="./src/jquery.min.js"></script>

            <script *src*="./src/script\_jquery.js"></script>

        </footer>

    </body>

</html>

Dentro do script\_jquery, usamos o seguinte código:

$(document).ready(function(){

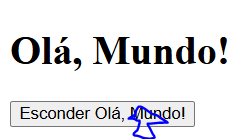
    $('button').click(function(){

        $('h1').hide();

    });

});

Para usar qualquer comando usa $ (cifrão/dollar), para pegar o documento html usa o objeto document. Então, começa por $(document).ready(); para verificar se a página foi carregada com sucesso e a partir disso, começar a imprimir alguma lógica. Supondo que queremos esconder o título h1, então dentro de $(document).ready();, cria uma função (function(){}) para escutar o clique do botão($(‘button’).click()), pega o elemento h1 e chama o método hide() (function(){$(‘h1’).hide()}. Lembrando, a estrutura fica como $(seletor).função();.





Se notarmos, os seletores que usados até o momento tão tags, então todos os botões **$(‘button’)**, todos os títulos **$(‘h1’)** vão ser afetados pelos métodos após o ponto **‘.’**. Entretanto, para aplicar a interação a um único elemento, podemos usar os ids pela **#** e por exemplo chamar o método **css** passando como parâmetro a propriedade “**color”** e como valor “red”.

.js

$(document).ready(function(){

    $('button').click(function(){

        $('h1').hide();

        $('#unico').css("color","red");

    });

});

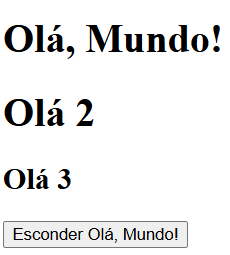
.html

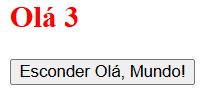
<h1> Olá, Mundo!</h1>

        <h1> Olá 2</h1>

        <h2 *id*="unico"> Olá 3</h1>

        <button>Esconder Olá, Mundo!</button>





Então para alterar o estilo dos elementos é utilizado o método **css** passando como parâmetro a propriedade e o valor do estilo entre aspas. Supondo que tenham duas mensagens (<p>) e dois botões (<button>) chamados azul e vermelho, ao clicar em um desses botões a cor do texto de cada mensagem é alterada na cor correspondente. Então, no html é importante, definir as mensagens com uma ‘class’ (msg) e os botões por #id (azul,vermelho), para que no Javascript possa ser possível escutar os eventos de clique de cada botão separadamente e aplicar a cor do texto nas mensagens usando a ‘class’.

.html

...

        <fieldset>

            <legend>Com Ids e Classes</legend>

            <p *class*="msg">Mensagem 1</p>

            <p *class*="msg">Mensagem 2</p>

            <button *id*="azul">Azul</button>

            <button *id*="vermelho">vermelho</button>

        </fieldset>

...

.js

$(document).ready(function () {

...

    $('#azul').click(function () {

        $('.msg').css("color", "blue");

    })

    $('#vermelho').click(function () {

        $('.msg').css("color", "red");

    })

});

Similarmente ao hide(), o método fadeOut() tem o efeito de esconder as mensagens, porém de maneira suave, e do contrário fadeIn() tem o efeito de exibir as mensagens também de maneira suave. Se colocar como parâmetro o termo ‘slow’, a transição será mais suave ainda. Para dar um tempo entre o fadeOut() e o fadeIn(), usa o método delay(3000) para atrasar ou pausar a execução do Javascript por 3s ou mais dependendo do valor inserido. De forma alternativa, pode trocar o termo ‘slow’ por ‘fast’ para acelerar um pouco mais a transição, e também definir os milissegundos dos efeitos fadeOut(3000) e fadeIn(5000);

  $('#azul').click(function () {

        $('.msg').css("color", "blue");

        $('.msg').fadeOut();

        $('.msg').delay(3000);

        $('.msg').fadeIn(5000);

    })

    $('#vermelho').click(function () {

        $('.msg').css("color", "red");

        $('.msg').fadeOut('slow');

        $('.msg').delay(3000);

        $('.msg').fadeIn('fast');

    })

Supondo que é preciso avisar o usuário que a cor foi alterada, este aviso pode ser feito usando a tag <span> no html para reservar o espaço do texto na página. No Javascript, a tag span é acessada pelo id #msg\_gravada usando principalmente o método ‘text’ para inserir o texto no conteúdo da tag, incrementando com cor de texto vermelho, borda vermelha, pausa de 3 segundos e esconder a tag.

.html

 <button *id*="vermelho">vermelho</button>

<span *id*="msg\_gravada"></span>

.js

$('#vermelho').click(function () {

        ...

        $('#msg\_gravada').text("Cor alterada com sucesso");

        $('#msg\_gravada').css("color", "red");

        $('#msg\_gravada').css("border", "1px solid red");

        $('#msg\_gravada').delay(3000);

        $('#msg\_gravada').fadeOut('fast');

    })

Um ponto importante é que podemos refatorar o código para deixá-lo mais legível, ao invés de para cada ação chamar por $(‘#msg/msg\_gravada/vermelho’), usar o $... apenas uma vez e concatenar as outras alterações por **‘.’** .

.js

 $('#azul').click(function () {

        $('.msg')

        .css("color", "blue")

        .fadeOut()

        .delay(3000)

        .fadeIn(5000);

    })

    $('#vermelho').click(function () {

        $('.msg')

        .css("color", "red")

        .fadeOut('slow')

        .delay(3000)

        .fadeIn('fast');

        $('#msg\_gravada')

        .text("Cor alterada com sucesso")

        .css("color", "red")

        .css("border", "1px solid red")

        .delay(3000)

        .fadeOut('fast');

    })

Uma segunda refatoração é colocar todas propriedades css como um único parâmetro no método css, para usar as propriedades e nomes compostos no padrão camel**C**ase dentro das {...}.

.js

...

$('#msg\_gravada')

        .text("Cor alterada com sucesso")

        .css({"color": "red", "border": "1px solid coral", "backgroundColor": "#F08080" })

        .delay(3000)

        .fadeOut('fast');

Supondo que ao invés de adicionar um fundo pelo método css, adicionar um fundo por uma classe, para usa o método .addClass(‘nome da classe’). Exemplo, no documento html insere duas classes green e red com as cores de fundo correspondente. No Javascript, chama o addClass(‘green’) pela $(‘#msg\_gravada’) para deixar o fundo verde no span.

.js

...

.css({"color": "red", "border": "1px solid red", "backgroundColor": "#F08080"})

        .addClass('green')

.html

<style>

*.green*{

                background-color: green;

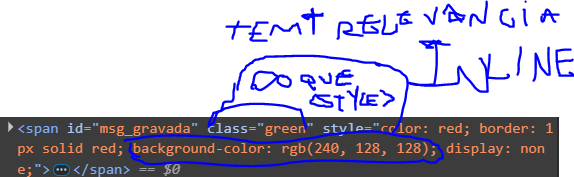
            }

*.red* {

                background-color: red;

            }

</style>





Porém, o background-color dentro do elemento está sobreescrevendo a classe green. Para resolver isso, é necessário remover o background-color do método css.

.js

...

.css({"color": "red", "border": "1px solid red"})

        .addClass('green')

.html





Dessa forma o elemento span carrega a class ‘green’ normalmente.

No botão vermelho, inclue a classe ‘red’ no elemento button para carregar o fundo vermelho. Uma vez clicado, o botão vai disparar a função e chamando o método removeClass(‘red’) para remover a classe e seu efeito.

.html

<style>

...

*.red* {

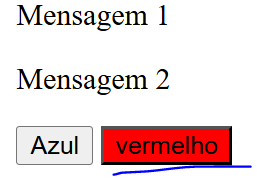
              background-color: red;

       }

    </style>

...

  <button *id*="vermelho" *class*="red">vermelho</button>



.js

$('#vermelho').removeClass('red');





Por fom, é criar um carrosel com uma seleção de imagens guiadas por links, tendo 4 imagens e 4 links, então quando clicar em um link uma imagem aparece e as demais são escondidas. Para isso são inseridas 4 imagens pela tag <img> com ids iNúmeroDaImagem, 4 links com a tag <a> com ids lNúmeroDoLink, e no style pegar todas as imagens exceto a imagem 1 e escondê-las.

.html

<style>

...

*#i2*,*#i3*,*#i4*{

                display: none;

            }

</style>

....

 <fieldset>

            <legend>Carousel</legend>

            <img *src*="./src/1.PNG" *alt*="imagem1" *id*="i1">

            <img *src*="./src/2.PNG" *alt*="imagem2" *id*="i2">

            <img *src*="./src/3.PNG" *alt*="imagem3" *id*="i3">

            <img *src*="./src/4.PNG" *alt*="imagem4" *id*="i4">

            <br>

            <a *href*="#" *id*="l1">1</a>

            <a *href*="#" *id*="l2">2</a>

            <a *href*="#" *id*="l3">3</a>

            <a *href*="#" *id*="l4">4</a>

 </fieldset>

.js

$('#l1').click(function(){

        $('#i1').show();

        $('#i2').hide();

        $('#i3').hide();

        $('#i4').hide();

    })

    $('#l2').click(function(){

        $('#i1').hide();

        $('#i2').show();

        $('#i3').hide();

        $('#i4').hide();

    })

$('#l3').click(function(){

        $('#i1').hide();

        $('#i2').hide();

        $('#i3').show();

        $('#i4').hide();

    })

$('#l4').click(function(){

        $('#i1').hide();

        $('#i2').hide();

        $('#i3').hide();

        $('#i4').show();

    })

