FIAP

GRADUAÇÃO

Javascript
Introdução

O que é Javascript

JavaScript é uma linguagem de programação interpretada. Foi originalmente implementada como parte dos navegadores web para que scripts pudessem ser executados do lado do cliente e interagissem com o usuário sem a necessidade deste script passar pelo servidor, controlando o navegador, realizando comunicação assíncrona e alterando o conteúdo do documento exibido.

É atualmente a principal linguagem para programação client-side em navegadores web. Começa também a ser bastante utilizada do lado do servidor através de ambientes como o node.js. Foi concebida para ser uma linguagem script com orientação a objetos baseada em protótipos, tipagem fraca, dinâmica e funções de primeira classe. Possui suporte à programação funcional e apresenta recursos como fechamentos e funções de alta ordem comumente indisponíveis em linguagens populares como Java e C++. É a linguagem de programação mais utilizada do mundo.

O que é Javascript

JAVASCRIPT É UMA LINGUAGEM DE SCRIPTS MAIS UTILIZADA NA INTERNET;

O PRINCIPAL OBJETIVO DA JAVASCRIPT É ADICIONAR INTERATIVIDADE NAS PÁGINAS.

NÃO É NECESSÁRIO COMPILADOR PARA EXECUTAR OS SCRIPTS.

JAVASCRIPT NÃO TEM NADA A VER COM O JAVA, EXCETO ALGUMAS SIMILARIDADES DE SINTAXE;

NÃO É NECESSÁRIO TIPAR AS VARIÁVEIS;

JAVASCRIPT É CASE SENSITIVE;

O que é Javascript

CLIENT-SIDE

COMPORTAMENTO

APRESENTAÇÃO

ESTRUTURA

O que fazer com Javascript

EVENTOS - “SÃO UTILIZADOS QUANDO ALGO ACONTECE NA PÁGINA”.

CLIQUE DO MOUSE;
CARREGAMENTO DA PÁGINA;
ENVIO DE INFORMAÇÕES.

LER E ESCREVER

É POSSIVEL ESCREVER EM CONTEÚDO DE ELEMENTOS HTML E DA MESMA FORMA LER OS SEUS VALORES.

VALIDAR DADOS

É POSSÍVEL VALIDAR INFORMAÇÕES ANTES MESMO DELAS SEREM ENVIADAS AO SERVIDOR.

Sintaxe e Formas de Utilização

SINTAXE - ELA É DEFINIDA PELA TAG HTML:

```
<script> ... </script>
```

UTILIZAÇÃO – EXISTEM 3 FORMAS DE UTILIZAÇÃO DO JAVASCRIPT:

- EM UM EVENTO INLINE;
- DIRETAMENTE NO CABEÇALHO OU CORPO DA PÁGINA;
- EM UM ARQUIVO EXTERNO.

Sintaxe e Formas de Utilização

EM UM EVENTO DA TAG HTML (INLINE):

<tag evento="instrução JS"></tag>

EX:

```
<button onclick="alert('Olá Mundo!!!')">Boas-vindas</button>
```

INTERNO NO CABEÇALHO OU CORPO DA PÁGINA

NO CABEÇALHO

```
<html>
    <head>
        <script>
            alert('Olá Mundo!!!')
        </script>
    </head>
    <body>
        .......
    </body>
</html>
```

NO CORPO

```
<html>
    <head>
        .......
    </head>
    <body>
        <script>
            alert('Olá Mundo!!!')
        </script>
    </body>
</html>
```

Sintaxe e Formas de Utilização

EM UM ARQUIVO JAVASCRIPT “EXTERNO”

ARQUIVO HTML

```html
<html>
  <head>
  </head>
  <body>
    <script type="text/javascript" src="js/script.js"></script>
  </body>
</html>
```

ARQUIVO JAVASCRIPT

```javascript
alert('Olá Mundo!!!');
```

Comentários no Código Javascript

EM JAVASCRIPT OS COMENTÁRIOS PODEM SER ADICIONADOS NO CÓDIGO DE DUAS MANEIRAS:

LINHA ÚNICA, REPRESENTADA POR DUAS BARRAS SEGUIDAS DO TEXTO

```
// UM EXEMPLO DE COMENTÁRIO NUMA LINHA ÚNICA
```

BLOCO, REPRESENTADO POR UM TEXTO ENTRE UMA BARRA E UM ASTERISCO

```
/*
   ESTE É UM COMENTÁRIO
   JAVASCRIPT
   EM BLOCO
*/
```

Sintaxe e Formas de Utilização

Como o javascript é uma linguagem de programação, precisamos de uma forma para vermos os resultados, ou seja, a saída do nosso código. Então vamos começar utilizando o método “write” que imprime o resultado no documento (nossa página).

EX:

```
document.write("Esta é a nossa primeira impressão na tela")
```

Temos também o método “console.log” que imprime o resultado no console do navegador.

EX:

```
console.log("Esta é a nossa primeira impressão no console")
```

Variáveis e Tipos de Dados

SÃO CARACTERISTICAS DAS VARIÁVEIS DO JAVASCRIPT

EM JAVASCRIPT AS VARIÁVEIS NÃO SÃO TIPADAS, LOGO BASTA DECLARÁ-LAS UTILIZANDO UMA DAS PALAVRAS RESERVADAS COMO “LET”;

O JAVASCRIPT É CASE SENSITIVE, OU SEJA, ELE FAZ DIFERENCIAÇÃO ENTRE LETRAS MAIÚSCULAS E MINÚSCULAS;

A FORMA DE DECLARAÇÃO DAS VARIÁVEIS UTILIZADA NO MERCADO É O CAMEL CASE, ONDE A PRIMEIRA PALAVRA COMEÇA COM LETRA MINÚSCULA E AS DEMAIS COM LETRA MAIÚSCULA SEM ESPAÇO ENTRE ELAS;

Variáveis e Tipos de Dados

É muito importante para usarmos uma linguagem de programação para resolver um problema conseguirmos armazenar os valores para manipulá-los. Para isso criamos pequenos espaços na memória que podem ser mutáveis ou imutáveis.
Vamos começar usando uma variável, espaço de memória mutável. Uma das formas de fazer isso é usando a palavra reservada "let" para declararmos, ou seja criamos uma variável.

EX:

```
let nome; //Cria uma variável com valor indefinido
let disciplina = null; //Cria uma variável vazia
let texto = "Este é um exemplo"; //Cria uma variável com um valor
```

Obs. Mais pra frente vamos conhecer outras opções

Variáveis e Tipos de Dados

O JAVASCRIPT PODE INTERPRETAR AS VARIÁVEIS COMO OS TIPOS:

INT = NÚMEROS INTEIROS .

EX:
```
let idadeAluno = 18
```

FLOAT = NÚMEROS FLUTUANTES COM CASAS DECIMAIS.

EX:
```
let valorProduto = 5.35
```

STRING = VARIÁVEIS DE TEXTO, ELAS DEVEM SER REPRESENTADAS ENTRE ASPAS SIMPLES OU DUPLAS.

EX:
```
let nomeAluno = "Pedro Henrique"
```

BOOLEANOS = QUE RECEBE OS VALORES FALSO OU VERDADEIRO.

EX:
```
let casado = true
```

ARRAY = CONJUNTO DE ELEMENTOS, DEVE SER REPRESENTADO ENTRE CONCHETES E OS VALORES SEPARADOS POR VIRGULA.

EX:
```
let linguagens = ["Java", "C#", "Python"]
```

Variáveis e Tipos de Dados

O JAVASCRIPT PODE INTERPRETAR AS VARIÁVEIS COMO OS TIPOS:

OBJETOS = Também podemos criar objetos com atributos e métodos

```
let carro = {
    cor: "preto",
    numPortas: 4,
    tipo: "sedan",
    modelo: "Onix",
    marca: "GM",
    correr: function(){
        alert("Estou correndo!!!")
    }
}

console.log(carro.modelo);
carro.correr()
```

Manipulando Textos e Números

PODEMOS PRECISAR MUDAR O TIPO DA VARIÁVEL, COMO POR EXEMPLO DE FLOAT PARA INTEGER, NESSE CASO USAMOS O "parseInt( )", O VALOR PERDERÁ AS CASAS DEPOIS DA VIRGULA.

EX:

```
let numFloat = 123.5432
console.log(parseInt(numFloat));
```

ELE IRÁ RETORNAR O VALOR COMO INTEIRO "123" .

SE A STRING FOR COMPOSTA DE NUMEROS PODEMOS TRANSFORMA-LA TAMBÉM, PARA ISSO PODEMOS USAR O "parseFloat( )", NO CASO DO VALOR TIVER CASAS DEPOIS DO PONTO.

EX:

```
let numString = "123.5432"
console.log(parseFloat(numString));
```

ELE IRÁ RETORNAR O VALOR COMO TIPO FLOAT "123.5432" .

Manipulando Textos e Números

CASO QUEIRA CONVERTER UMA VARIÁVEL PARA STRING DEVEMOS USAR O MÉTODO "toString( )".

EX.

```
let numFloat = 123.5432
console.log(numFloat.toString());
```

OBS. O RESULTADO SERÁ IGUAL AO ORIGINAL, SÓ QUE EM FORMATO STRING.

PARA CONFERIR A CONVERSÃO DOS VALORES, PODEMOS UTILIZAR O VERIFICADOR "typeof", QUE RETORNARÁ O TIPO DA VARIÁVEL.

EX.

```
let numFloat = 123.5432
let numString = numFloat.toString()
let verificar = typeof numString
console.log(verificar);
```

Manipulando Textos e Números

OS TEXTOS (STRING) DEVEM SEMPRE ESTAR ENTRE ASPAS DUPLAS OU SIMPLES.

EX:

```
let nomeAluno1 = "Pedro Henrique"

let nomeAluno2 = 'Luiz Augusto'
```

QUANDO PRECISAMOS USAR ALGUM CARATER RESERVADO EM NOSSO TEXTO USAMOS A BARRA INVERTIDA "\" ANTES DO CARACTERE PARA ELE FAZER PARTE DO TEXTO.

EX:

```
let pedido = 'Quero um copo d\'água!'
```

PODEMOS VERIFICAR O TAMANHO DA NOSSA STRING UTILIZANDO O MÉTODO "length"

EX:

```
let pedido = 'Quero um copo d\'água!'
alert(pedido.length)
```

Manipulando Textos e Números

OUTRO MÉTODO QUE PODEMOS UTILIZAR É O “indexOf( )”, ELE RETORNA A PRIMEIRA POSIÇÃO NA STRING DE UM TRECHO DO TEXTO.

EX:

```
let txt = "Estão chegando as provas"
console.log(txt.indexOf("as"));
```

ELE IRÁ RETORNAR O VALOR “15” QUE É A POSIÇÃO DA PRIMEIRA OCORRÊNCIA.

PARA SABERMOS A ÚLTIMA VEZ QUE O TRECHO APARECE NO TEXTO UTILIZAMOS O “lastIndexOf( )”.

EX:

```
let txt = "Estão chegando as provas"
console.log(txt.lastIndexOf("as"));
```

ELE IRÁ RETORNAR O VALOR “22” QUE É A POSIÇÃO DA ÚLTIMA OCORRÊNCIA.

- 18/10/2023

Manipulando Textos e Números

NÓS PODEMOS TAMBÉM EXTRAIR PARTE DO TEXTO, PARA ISSO UTILIZAMOS O "slice( )", QUANDO APONTAMOS O INÍCIO E O FIM DO TRECHO QUE QUEREMOS EXTRAIR E ELE NOS RETORNA O TRECHO COMO UMA NOVA STRING.

EX:

```
let txt = "Estão chegando as provas"
console.log(txt.slice(0,5));
```

ELE IRÁ RETORNAR A STRING "Estão" QUE É O TRECHO DO TEXTO DE 0 A 5.

TEMOS TAMBÉM O "substr( )" QUE RECEBE A POSIÇÃO INICIAL E A QUANTIDADE DE CARACTERES QUE QUEREMOS PEGAR.

EX:

```
let txt = "Estão chegando as provas"
console.log(txt.substr(6,8));
```

ELE IRÁ RETORNAR A STRING "chegando" QUE AS 8 POSIÇÕES DEPOIS DA 6ª.

Manipulando Textos e Números

É POSSÍVEL TAMBÉM SUBSTITUIRMOS UM TRECHO DO TEXTO O “replace( )”, RECEBE O TRECHO QUE DEVE SER SUBSTITUIDO E O TRECHO QUE ENTRARÁ NO LUGAR DELE.

EX:

```
let txt = "Estão chegando as provas"
console.log(txt.replace("provas","avaliações"));
```

ELE IRÁ RETORNAR A FRASE “Estão chegando as avaliações!” .

O MÉTODO “toUpperCase( )” IRÁ CONVERTER TODA A STRING EM LETRAS MAIÚSCULAS.

EX:

```
let txt = "Estão chegando as provas"
console.log(txt.toUpperCase);
```

O MÉTODO “toLowerCase( )” IRÁ CONVERTER TODA A STRING EM LETRAS MINÚSCULAS.

EX:

```
let txt = "Estão chegando as provas"
console.log(txt.toLowerCase);
```

Manipulando Textos e Números

PARA OS NÚMEROS NÓS TAMBÉM PODEMOS DEFINIR ALGUMAS FORMAS DE APRESENTAÇÃO, POR EXEMPLO TEMOS O “toFixed( )”, QUE DEFINE O NÚMERO DE CASAS DECIMAIS DO VALOR.

EX:

```
let num = 123.5432
console.log(num.toFixed(2))
```

ELE IRÁ RETORNAR O VALOR COM 2 CASAS DECIMAIS “123.54” .

SE QUISERMOS DEFINIR UM NÚMERO DE CASAS INDEPENDENTE DO PONTO USAMOS O “toPrecision( )”, QUE DEFINE O NÚMERO DE CASAS DO VALOR COMO UM TODO.

EX:

```
let num = 123.5432
console.log(num.toPrecision(4));
```

ELE IRÁ RETORNAR O VALOR COM 4 DIGITOS “123.5” .

Manipulando Textos e Números

Operadores matemáticos

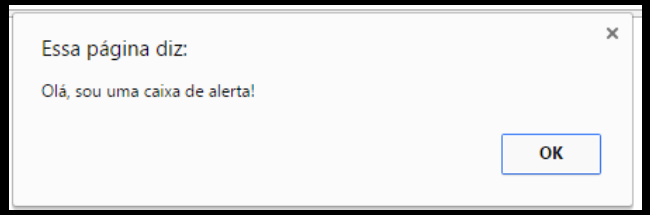
Para manipulamos os valores podemos utilizar os operadores matemáticos:

+	SOMA
-	SUBTRAÇÃO
/	DIVISÃO
*	MULTIPLICAÇÃO
%	MÓDULO
**	POTÊNCIA

Caixas de Mensagem

Em JS as caixas de mensagens são usadas para apresentar informações para o usuário. São elas:

CAIXA DE ALERTA:

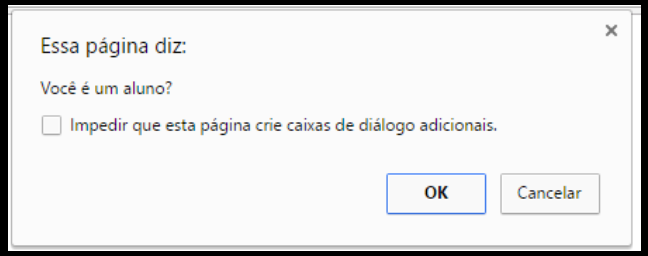


EX:

```
alert("Olá, sou uma caixa de alerta!");
```

Caixas de Mensagem

CAIXA DE CONFIRMAÇÃO:



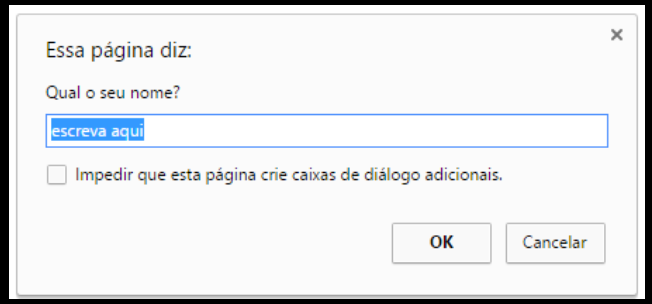
ELA RETORNA UM VALOR BOLEANO, TRUE PARA OK E FALSE PARA CANCELAR

EX:

```
let teste = confirm("você é um aluno?")
console.log("Resultado da caixa confirm: ", teste);
```

Caixas de Mensagem

CAIXA DE TEXTO:



ELA RETORNA A STRING DIGITADA PARA OK E NULL PARA CANCELAR.

EX:

```
let texto = prompt("Qual o seu nome?","escreva aqui")
console.log("O nome dele é ", texto)
```

OBS. SE QUISER, SEPARANDO POR UMA VIRGULA VOCÊ PODE INSERIR UMA MENSAGEM QUE FICA SELECIONADA DENTRO DO CAMPO.

Estruturas de Decisão e Repetição

IF “SE”

- A estrutura mais usada e mais importante é a estrutura IF (Se), usamos ela para verificar se uma afirmação é verdadeira ou falsa, ou seja ela nos retorna um valor boleano.

Ex:

```
if(true){
    "INTRUÇÕES QUE SERÃO REALIZADAS CASO SEJA VERDADEIRO"
}
```

O comando **IF** verifica o **teste lógico** que está entre parênteses e se for verdadeiro executa as **instruções** que estão dentro das Chaves.

IF “SE”

- Mas se para podermos saber se algo é verdadeiro ou falso precisamos fazer um teste lógico, como podemos fazer isso?

  Um teste lógico sempre vai nos devolver as respostas “Falso” ou “Veradeiro” e para isso podemos contar com os operadores lógicos, são eles:

Operador	Nome	Exemplo
>	Maior	10 > 7
<	Menor	7 < 10
>=	Maior ou igual	15 >= 15
<=	Menor ou igual	8 <= 8
!=	Diferente	4 != 3
==	Igual	5 == 5

IF “SE”

Vamos fazer um teste, digite o código abaixo:

```
if(10 > 7){
    document.write("Dez é maior que sete!!!")
}
```

Aqui verificamos se 10 é maior do que 7, como esta afirmação é verdadeira ele irá imprimir a frase na tela. Podemos também usar variáveis nos teste, veja como ficaria:

```
let valor = 10
if(valor > 7){
    document.write("O valor é maior que sete!!!")
}
```

IF “SE” – ELSE “SENÃO”

- Você reparou que nossa estrutura lógica só realizava uma ação se o resultado fosse verdadeiro, se fosse falso nada acontecia. Na grande maioria das vezes precisamos também dar instruções caso aquele teste nos retorna falso, para isso usamos um complemento no IF chamado ELSE, senão em português. Vamos fazer um teste:

```
let valor = 10
if(valor >= 15){
    document.write("O valor é maior ou igual a 15")
}else{
    document.write("O valor é menor que 15")
}
```

Reparem que é como se estivéssemos conversando com a máquina: Se o valor for maior ou igual a 15 escreva no documento: O valor é maior ou igual a 15. Senão escreva: O valor é menor que 15.

IF “SE” – ELSE IF “SENÃO SE” - ELSE “SENÃO”

- Podemos também fazer uma serie de verificações para chegar na saída desejada, para isso também podemos contar com o ELSE IF “senão se”, após o 1º IF, exemplo:

```
let idade = 19
if(idade <= 12){
    document.write("É uma criança")
}else if(idade <= 17){
    document.write("É um adolescente")
}else if(idade <= 29){
    document.write("É um jovem")
}else{
    document.write("É um adulto")
}
```

Devemos tomar cuidado com a criação destas estruturas pois se as verificações estiverem na ordem errada ela são irá trazer o valor esperado.

&& “e” - || “ou”

- Ainda falando sobre a estrutura de decisão IF, podemos ter mais de um teste lógico em cada interação, mas para isso precisamos avisar se todos os testes dever ser verdadeiros ou se só um já basta. Para isso usamos os operadores “&& para e” e “|| para ou”.

- Ex: Se para ser válido o numero deve ser maior que 10 **e** menor que 20:

```
let valor = 15
if(valor > 10 && valor < 20){
    document.write("Este valor é válido")
}
```

&& “e” - || “ou”

Agora um exemplo usando o || “ou”.

Ex: Se para ser válido o numero deve ser menor que 10 **ou** maior que 20:

```
let valor = 25
if(valor < 10 || valor > 20){
    document.write("Este valor não está entre 10 e 20")
}
```

Switch

O Switch é uma estrutura de decisão que não trabalha com condições “verdadeiros e falsos”, mas sim com valores diretos.

EX:

```
let teste = 2
switch (teste) {
    case 1:
        document.write("Foi de primeira!")
        break;
    case 2:
        document.write("Foi na segunda vez!")
        break;
    default:
        document.write("Não foi nem na primeira e nem na segunda!")
        break;
}
```

Estruturas de Repetição

Na programação muitas vezes precisamos repetir ações diversas vezes para obter o resultado esperado. Por exemplo, se temos que listar os dados de um banco, verificar os itens de uma lista, ou seja temos inúmeras razões para precisar de repetições quando falamos em lógica. Para isso usamos as estruturas de repetição.

INCREMENTO e DECREMENTO

Antes de entrarmos nos comandos de repetição, vamos entender uma parte fundamental para realizarmos estas repetições, o incremento e decremento.

Vamos fazer um exemplo de incremento:

```
let valor = 7
console.log(valor);  //valor igual a 1
valor = valor+1
console.log(valor);  // valor igual a 8
```

Repare que usamos a própria variável para incrementar mais 1 ao seu valor.

INCREMENTO e DECREMENTO

Mas dá para facilitar ainda mais este código, veja outra forma:

```
let valor = 7
console.log(valor);  //valor igual a 1
valor = valor+1
console.log(valor);  // valor igual a 8
valor++
console.log(valor);  // valor igual a 9
```

Quando usamos “++” após a variável, estamos indicando que deve ser incrementado mais 1 ao seu valor.

INCREMENTO e DECREMENTO

Bem, se se precisarmos incrementar mais de um, como podemos fazer?

```
let valor = 7
console.log(valor)  //valor igual a 1
valor += 3  // valor igual a 10
```

Quando usamos “+=” estamos dizendo que a partir daquele momento a variável valor igual a seu próprio valor + 3.

INCREMENTO e DECREMENTO

Em alguns casos pode ser que precisemos fazer o contrário, como uma contagem regressiva, por exemplo. Para estes casos, da mesma forma que usamos **++** para incrementar, podemos usar **--** para decrementar um valor.

```
let valor = 7
console.log(valor)  // valor igual a 7
valor--
console.log(valor)  // valor igual a 6
```

Quando usamos “--” após a variável, estamos indicando que deve ser decrementado menos 1 ao seu valor.

INCREMENTO e DECREMENTO

- Bem, se se precisarmos decrementar mais de um? Faremos da mesma forma que o incremento:

```
let valor = 8
console.log(valor)  // valor igual a 8
valor -= 4
console.log(valor)  // valor igual a 4
```

Quando usamos “-=” estamos dizendo que a partir daquele momento a variável valor igual a seu próprio valor - 4.

INCREMENTO e DECREMENTO

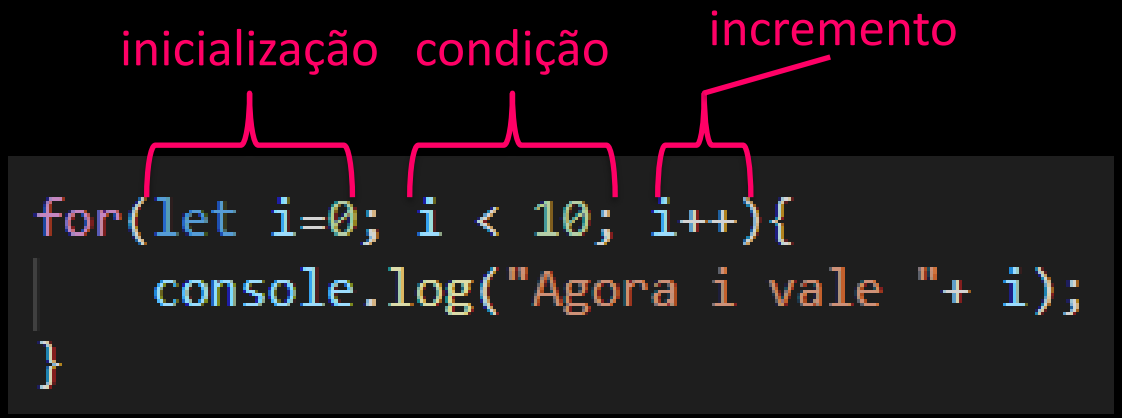
- Usando a mesma lógica do += e do -=, podemos também usar este artifício para multiplicação e divisão (***=** e **/=**):

```
let valor = 5
console.log(valor) //valor igual a 5
valor *= 4
console.log(valor); //valor igual a 20
valor /= 8
console.log(valor); //valor igual a 2.5
```

As operações ** e // não são válidas, pois se funcionassem não iriam mudar o valor afinal de contas!

FOR “PARA”

O comando FOR é a estrutura mais utilizada, e procura utilizar criar repetições de instruções pelo número de vezes que precisarmos. Vamos ver como ela funciona:



FOR “PARA”

Podemos usar ele também para percorrer um array, por exemplo:

```
let frutas = ["maça","banana","uva","pera"]
for(let i=0; i < frutas.length; i++){
    console.log(frutas[i]);
}
```

Neste exemplo usamos a propriedade **length** para saber o tamanho do array.

FOR OF

Usando o exemplo anterior vamos ver uma maneira mais leve de percorrer um array:

```
let frutas = ["maça","banana","uva","pera"]
for (let fruta of frutas) {
    console.log(fruta);
}
```

O **for of** é utilizado para percorrer um array, podemos criar uma variável que automaticamente recebe os valores na interação, neste exemplo é a “fruta”.

WHILE - “ENQUANTO”

Para fazer repetições no código também podemos utilizar o comando “while”, a sintaxe dele é diferente do for, por exemplo temos que ter uma variável antes de inicializa-lo:

```
let cont = 0
while (cont < 10){
    console.log("passagem "+cont);
    cont++
}
```

Repare que ele necessita dos mesmos 3 elementos que o FOR para funcionar “iniciar uma variável”, uma “condição” e um “incremento”, eles só estão em locais diferentes.

Importante: se não houver um incremento na variável ele pode ficar em loop infinito!!!

DO WHILE - “FAZER ENQUANTO”

Existem também uma opção quando precisamos que mesmo que a condição já esteja atingida ou ultrapassada, ele realize pelo menos uma vez a instrução:

```
let cont = 0
do{
    console.log("Passagem "+cont)
    cont++
}while(cont == 0)
```

cont é igual a 0
somente na primeira passagem

```
let cont = 0
do{
    console.log("Passagem "+cont)
    cont++
}while(cont < 0)
```

cont não é menor que 0
em outras estruturas
nem passaria a 1ª vez.

Funções

- Como o nome já diz, o javascript é um script que carrega quando a página é aberta. Sendo assim, precisamos de uma forma para controlar o momento e as condições para estas instruções sejam executadas. Para resolver este problema utilizamos as funções, com elas podemos controlar a execução das instruções.

- Exemplo de função recebendo argumentos:

```
function nomeFuncao(arg1, arg2){
    return arg1 + arg2
}

console.log(nomeFuncao(5,4));
```

Funções

Exemplo de função sem argumentos:

```
function avisar(){
    alert("Esta é uma função!")
}

avisar()
```

DOM

O DOM é um modelo de documento carregado pelo navegador. Este documento é representado através de uma árvore de nós, onde cada um destes nós representa uma parte do documento (por ex. um elemento, texto ou comentário). O DOM define um padrão para acessar documentos HTML.

O DOM é uma das APIs mais usadas na Web porque ele permite que cada código rodando no navegador acesse e interaja com cada nó do documento. Os nós podem ser criados, movidos ou modificados. Listeners de evento podem também ser adicionados aos nós para serem disparados quando um dado evento ocorrer.

DOM

- Podemos acessar e manipular as informações e atributos dos nossos elementos HTML pelo ID, pela Classe, pelo name e pelo nome da TAG. Sendo que a opção mais usada é pelo **ID** (por ser único). Os demais usamos quando queremos percorrer ou coletar várias informações.

Usando o ID: Para acessarmos o ID usamos a sintaxe:

```
document.getElementById("nome").innerHTML = "Luís Carlos"
```

document se refere a página html;
getElementById( ) recupera um elemento pelo seu ID;
innerHTML utilizamos para alterar/inserir o conteúdo do elemento (Tag) de marcação.

DOM

Exemplo:

```
<!-- EXEMPLO getElementByID -->

<h1 id="titulo">Exemplo por ID</h1>

Nome: <input type="text" name="" id="idNome">
<button onclick="mudar()">Clique Aqui</button>

<script>
    let novo = document.getElementById("idNome")

    function mudar(){
        document.getElementById("titulo").innerHTML = novo.value
    }
</script>
```

DOM

Usando a Classe: Acessamos utilizando a classe quando precisamos retornar uma coleção de elementos com uma mesma classe :

```
document.getElementsByClassName("noticia")[0].value
```

document se refere a página html;
getElementsByTagName(“classe”)[0] recupera um elemento com a classe especificada pela sua posição, começando por “0”;
value retorna o valor “conteúdo” do elemento.

DOM

- Exemplo:

```
<h2>Exemplo por ClassName</h2>

<p>Primeiro: <span class="corredor"></span></p>
<p>Segundo : <span class="corredor"></span></p>
<p>Terceiro: <span class="corredor"></span></p>
<br>
Novo Classificado: <input type="text" id="novo"> <br>
Posição: <input type="number" id="posicao" min="1" max="3">
<button onclick="inserir()">Inserir</button>

<script>
    function inserir(){
        let numero = document.getElementById("posicao").value -1
        let novo = document.getElementById("novo").value
        document.getElementsByClassName("corredor")[numero].innerHTML = novo
    }
</script>
```

DOM

- **Usando o TagName:** Assim como com o name, podemos acessar elementos pelo nome da Tag se precisarmos retornar uma coleção de elementos dessa tag :

```
document.getElementsByTagName("li")[0].value
```

document se refere a página html;
getElementsByTagName(“tag”)[0] recupera um elemento pelo nome da tag especificado pela sua posição, começando por “0”;
value retorna o valor “conteúdo” do elemento.

DOM

- Exemplo:

```
<h2>Exemplo por TagName</h2>

<ul>
    <li>Janeiro</li>
    <li>Fevereiro</li>
    <li>Março</li>
    <li>Abril</li>
    <li>Maio</li>
</ul>

Escolha um mês: <input type="number" id="numero" min="1" max="5">
<button onclick="mostrar()">Mostrar</button>
<p>Mês Escolhido: <span id="resultado"></span></p>

<script>
    function mostrar(){
        let num = document.getElementById("numero").value -1
        let mes = document.getElementsByTagName("li")[num].innerHTML;
        document.getElementById("resultado").innerHTML = mes
    }
</script>
```

DOM

Também temos uma outra forma de selecionar os elementos da página, podemos usar o **querySelector( )**, que diferente dos anteriores recebe um seletor CSS para identificar o elemento.

EX:

```
document.querySelector("#nome")
```

OBS. Ele só retorna a 1ª ocorrência.

DOM

Quando temos que pegar um elemento pela classe ou Tag e definir qual das ocorrências queremos, no caso de mais de uma, podemos usar o **querySelectorAll( )** .
EX:

```
document.querySelectorAll(".btn")[0]
```

OBS. Aqui usamos os colchetes para colocar a posição que queremos selecionar.

DÚVIDAS?

“A dúvida é o princípio da sabedoria.”

- Aristóteles


FIAP