



Universidade Federal de Uberlândia
Faculdade de Computação

Programação para Internet

Módulo 4

Introdução ao JavaScript e DOM
(*Document Object Model*)

Prof. Dr. Daniel A. Furtado

Introdução

Por que estudar JavaScript?

JavaScript é uma das três linguagens Web que todo desenvolvedor deveria aprender:

1. **HTML** para definir o *conteúdo* de páginas Web;
2. **CSS** para especificar o *layout* e a formatação das páginas Web;
3. **JavaScript** para programar o *comportamento* das páginas Web.

Ref: adaptado de *w3schools.com*

Por que estudar JavaScript?

- **JavaScript** é bastante utilizada em situações que exigem dinamismo e respostas imediatas do usuário. É possível, por exemplo:
 - Alterar o conteúdo e o layout da página Web em tempo de exibição;
 - Executar ações para inicializar a página Web assim que ela é carregada;
 - Executar ações de encerramento quando a página é fechada;
 - Executar operações em resposta a ações do usuário, como “clicar em um botão” ou “selecionar uma opção”;
 - Validar o conteúdo de campos de formulários à medida que o usuário os preenche;
- Entretanto, questões relacionadas à segurança da aplicação, como validação efetiva dos formulários e validação de usuários, devem ser implementadas em linguagem no lado do servidor, como PHP, ASP, etc.

Linguagem JavaScript

- **JavaScript** é uma linguagem de *script* orientada a objetos utilizada para desenvolvimento de aplicações Web;
- É comumente referenciada como **JS**;
- Originalmente foi denominada *Mocha*, depois *LiveScript* e então JavaScript após acordo de licença realizado entre a Netscape e a Sun em 1995;
- Os scripts em **JavaScript** são normalmente executados no lado do **cliente**, ou seja, no programa **navegador** do usuário (como o Google Chrome, o Microsoft Edge, etc.);
- Os scripts em **JavaScript** são interpretados pelo navegador. Não é necessário realizar uma compilação explícita do código;
- Não confundir com a linguagem de programação Java:

JavaScript não é nenhuma extensão ou adaptação da linguagem Java

JavaScript - Motivação

- **JavaScript** pode mudar o **conteúdo** de elementos HTML;
- Neste exemplo o conteúdo do parágrafo é alterado quando o botão é pressionado (ver [Anexos/JavaScript-Exemplo01.html](#));
- *`document.getElementById("idDoElemento")`* possibilita resgatar um objeto JavaScript correspondente ao elemento HTML com tal **id**;
- *`innerHTML`* é uma propriedade do objeto que dá acesso ao conteúdo do elemento HTML;

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>
<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick="document.getElementById('demo').innerHTML =
'Hello JavaScript!'">Click Me!</button>

</body>
</html>
```

Adaptado de [w3schools.com](#)

JavaScript - Motivação

JavaScript pode mudar o valor dos **atributos** dos elementos HTML

```
<!DOCTYPE html>
<html>
<body>

<h2>O que JavaScript pode fazer?</h2>
<p>JavaScript pode alterar atributos de elementos HTML</p>
<p>Neste caso JavaScript altera o valor do atributo 'src' da imagem.</p>

<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">
Ligar a Luz </button>



<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">
Desligar a luz</button>

</body>
</html>
```



Adaptado de w3schools.com

JavaScript - Motivação

- **JavaScript** pode mudar o **estilo CSS** dos elementos HTML
- Neste exemplo JavaScript é utilizada para alterar o valor da propriedade CSS “*font-size*” do parágrafo quando o botão é pressionado (ver [Anexos/JavaScript-Exemplo03.html](#));

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<button type="button"
onclick="document.getElementById('demo').style.fontSize='35px'">
Click Me!</button>

</body>
</html>
```

Adaptado de [w3schools.com](#)

JavaScript - Motivação

- **JavaScript** pode **ocultar e exibir** os elementos HTML
- Neste exemplo JavaScript é utilizada para alterar a propriedade CSS “*display*” da imagem para o valor “none”, causando o desaparecimento da imagem (ver [Anexos/JavaScript-Exemplo04.html](#)).

```
<!DOCTYPE html>
<html>
<body>

<h2>O que JavaScript pode fazer?</h2>
<p>JavaScript pode ocultar e exibir elementos HTML</p>

<button onclick="document.getElementById('myImage').style.display='none'">
Ocultar luz</button>

<button onclick="document.getElementById('myImage').style.display='block'">
Mostrar luz</button>



</body>
</html>
```

Atributos de Eventos da Linguagem HTML

- O atributo *onclick* utilizado nos exemplos anteriores é denominado *atributo de evento* da linguagem HTML;
- Um atributo de evento possibilita associar uma ação que deve ser executada quando um evento ocorrer (no caso anterior, o evento ‘clicar’ sobre o elemento)
- Alguns outros atributos de eventos frequentemente utilizados:
 - *onMouseEnter*. Quando o usuário ‘entra’ com o ponteiro do mouse sobre o elemento;
 - *onMouseLeave*. Quando o usuário ‘retira’ o ponteiro do mouse do elemento;
 - *onMouseDown*. Quando o usuário pressiona um botão do mouse sobre o elemento;
 - *onMouseUp*. Quando o usuário solta o botão do mouse sobre o elemento;
 - *onMouseOver*. Quando o usuário move o ponteiro do mouse sobre o elemento;
 - *onChange*. Em alguns elementos, quando o seu valor (*value*) muda;
 - *onFocus*. Quando o elemento recebe foco.

Código JavaScript e HTML

- Nos exemplos anteriores inserimos código JavaScript *inline*, ou seja, diretamente dentro de um elemento HTML (como valor do atributo *onclick*);
- Também é possível inserir código JavaScript dentro do documento HTML utilizando a TAG `<script>`. O trecho de código pode ser inserido dentro do cabeçalho (`<head>`) ou do corpo `<body>` da página:

```
<script>  
    // coloque aqui o código JavaScript (HTML5)  
</script>
```

- Ver [Anexos/JavaScript-Exemplo05.html](#)
- Para inserir código JavaScript em arquivo externo, deve-se fazer referência ao arquivo dentro do documento HTML:

```
<script src="arquivoComCodigoJavaScript.js"></script>
```

Utilize letras minúsculas em nomes de arquivos

- A maioria dos servidores Web (Apache, Linux) é sensível à caixa (*case sensitive*) quanto aos nomes de arquivos:
 - ***image.jpg*** não pode ser acessado como ***Image.jpg***.
- Outros servidores Web (Microsoft IIS) não são sensíveis à caixa:
 - ***image.jpg*** pode ser acessado como ***Image.jpg*** ou ***image.jpg***
- Para evitar problemas, utilize sempre letras minúsculas nos nomes de arquivos (se possível).

Entrada e Saída de Dados

Saída de Dados

JavaScript pode gerar e mostrar dados de diferentes maneiras:

- Escrevendo no conteúdo de um elemento HTML por meio da propriedade *innerHTML* (há propriedades similares como *innerText* e *textContent* – pesquisar)
- Escrevendo no documento HTML propriamente dito usando *document.write('conteudo');*
- Escrevendo em uma caixa de mensagens usando *window.alert('mensagem');*
- Escrevendo no *console* do navegador, para fins de desenvolvimento/debug, usando *console.log('info');*

Saída de Dados - Exemplo

```
<!DOCTYPE html>
<html>
<body>

<h1>Linguagem JavaScript</h1>
<p>Utilizando document.write para gerar conteudo</p>

<script>
    console.log('Escrevendo no console do navegador...');
    for (var i = 0; i < 10; i++)
        document.write('Texto gerado por funcao JavaScript <br>');
</script>

<button type="button" onclick="window.alert('Obrigado!')">Clique
aqui!</button>

</body>
</html>
```

- **window, document e console** são objetos;
- **alert, write e log** são métodos dos respectivos objetos;
- Para exibição do console do navegador, tecle <F12> e procure pela aba *console*.

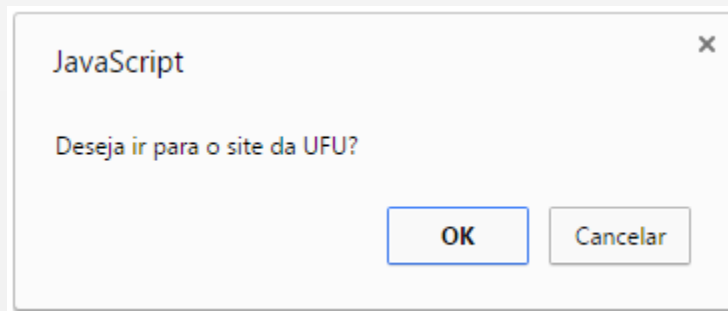
Caixa de Diálogo de Confirmação - *confirm*

- O método *confirm* apresenta uma caixa de diálogo solicitando uma confirmação do usuário, geralmente com os botões “Ok” e “Cancelar”;
- Devolve **verdadeiro** quando o botão “Ok” é pressionado; ou **falso**, caso o usuário feche a janela ou pressione “Cancelar”;

```
...
```

```
if (confirm('Deseja ir para o site da UFU?'))  
    window.location = "http://www.ufu.br";
```

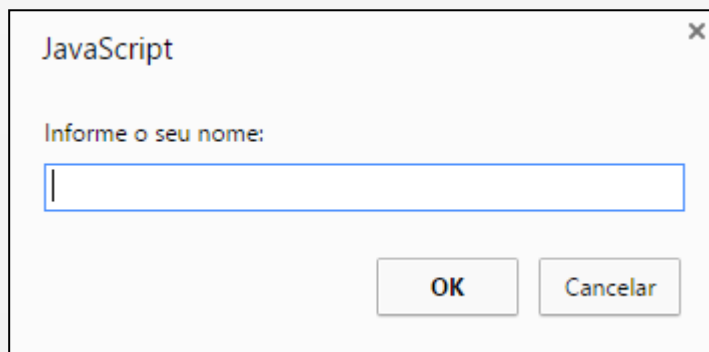
```
...
```



Caixa de Diálogo de Entrada - *prompt*

- O método *prompt* apresenta uma caixa de diálogo com um campo para preenchimento;
- Retorna uma **string** correspondente ao texto informado no campo; ou *null* caso o usuário clique no botão “Cancelar”;

```
...  
  
var nome = prompt('Informe o seu nome:');  
if (nome != null)  
    alert('Bem vindo, ' + nome);  
...
```



Caixa de Diálogo de Entrada - *prompt*

- O valor retornado pela função *prompt* é sempre do tipo **string** (exceto o valor *null*)
- Assim, caso seja fornecido um número inteiro no campo de texto, deve-se fazer a conversão para **inteiro** antes de processar o número (por exemplo, utilizando a função **parseInt**). Exemplo:

```
...  
  
var str = prompt('Informe um número inteiro:');  
if (str != null)  
{  
    x = parseInt(str);  
    dobro = 2 * x;  
    alert('O dobro do numero eh: ' + dobro);  
}  
...
```

- Similarmente, a função **parseFloat** faz a conversão de **string** para **float**;

A Linguagem JavaScript

Código JavaScript e HTML

- JavaScript é *case sensitive*;
- Declarações em JavaScript devem terminar com o *ponto-e-vírgula*;
- Comentários de linha são do tipo *// comentário*
- Comentários de bloco são do tipo */* comentário */*

Variáveis

- Variáveis podem ser declaradas com ou sem a palavra **var**;
- O tipo da variável é determinado automaticamente;

```
<!DOCTYPE html>
<html>
<body>

<h1>Definindo variaveis - Tecle F12 para depurar</h1>

<script>

    var a = 1; b = 2; c = 3;
    var delta = b*b - 4*a*c;
    document.write('O valor do discriminante eh: ' + delta);
    var str = 'Programação para Internet'; // str é uma string
    var A = true; // A é uma variável booleana

</script>

</body>
</html>
```

Estruturas Condicionais e de Repetição

```
if (expressão) {  
    // operações  
}
```

```
if (expressão) {  
    // operações  
}  
else {  
    // operações  
}
```

```
if (expressao1) {  
    // operações 1  
}  
else if (expressao2) {  
    // operações 2  
}  
else {  
    // operações 4  
}
```

```
for (var i = 0; i < 10; i++)  
{  
    // operações  
    // operações  
}
```

```
while (expressao)  
{  
    // operações  
    // operações  
}
```

```
do {  
    // operações  
    // operações  
} while (expressao)
```

```
switch (expressao) {  
    case condicao1:  
        // operações  
        break;  
    case condicao1:  
        // operações  
        break;  
        ...  
    case condicaoN:  
        // operações  
        break;  
    default:  
        // operações  
}
```

Operadores Aritméticos, Relacionais e Lógicos

Operadores Aritméticos

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira (módulo)
++	Incremento
--	Decremento

Operadores Relacionais e Lógicos

Operador	Significado
==	Comparação por igualdade
===	Comparação por igualdade, incluindo valor e tipo
!=	Diferente
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a
&&	“E” lógico
	“Ou” lógico
!	Negação lógica

OBS: O Operador **+** também pode ser utilizado para concatenar *strings*

Operador de Adição e Concatenação

- Deve-se utilizar o operador + com atenção, pois a operação executada (soma ou concatenação) depende do tipo dos operandos;
- JavaScript avalia as expressões da esquerda para a direita;
- Ao avaliar um **par de operandos**, se um deles for **string** e o outro **numérico**, então o numérico será convertido para string.
- Exemplos

```
x = 5 + 5;           // x terá o valor 10
y = "5" + 5;         // y terá a string '55'
z = "Hello" + 5;     // z terá a string 'Hello5'
w = 2 + 4 + "5";     // w terá a string '65'
p = "5" + 2 + 4;     // p terá a string '524'
```


Diferença dos Operadores `==` e `===`

- O operador `==` compara apenas os valores dos operandos. Se os operandos forem de tipos diferentes, uma conversão é realizada e os valores convertidos são comparados;
- O operador `===` compara o valor e o tipo dos operandos. A comparação de operandos de tipos diferentes sempre resulta em *falso*.
- Exemplos:

```
• 1 == "1";           // true
  1 == true;          // true

• 1 === "1";          // false
  1 === true;          // false

• var x = 10;
  var y = "10";
  x == y;              // true

• var x = 10;
  var y = "10";
  x === y;             // false
```

Operadores de Atribuição

Operador	Significado	Exemplo
=	Atribuição	<pre>var x = 0; // atribui o valor 0 a x</pre>
+=	Atribuição com soma	<pre>var x += y; // equivalente a: x = x + y</pre>
-=	Atribuição com subtração	<pre>var x -= y; // equivalente a: x = x - y</pre>
*=	Atribuição com multiplicação	<pre>var x *= y; // equivalente a: x = x * y</pre>
/=	Atribuição com divisão	<pre>var x /= y; // equivalente a: x = x / y</pre>
%=	Atribuição com módulo	<pre>var x %= y; // equivalente a: x = x % y</pre>

Outros Cálculos Matemáticos

O objeto **Math** disponibiliza uma série de métodos para cálculos matemáticos. Alguns deles são:

Método	Descrição
Math.sqrt(x)	Retorna a raiz quadrada de x.
Math.pow(x,y)	Retorna o valor de x elevado a y.
Math.PI	Retorna o valor da constante matemática PI.
Math.sin(x)	Calcula o seno do angulo x, dado em radianos.
Math.cos(x)	Calcula o cosseno do angulo x, dado em radianos.
Math.tan(x)	Calcula a tangente do angulo x, dado em radianos.
Math.round(x)	Retorna o valor de x arredondado para o inteiro mais próximo.
Math.random()	Retorna um número fracionário aleatório entre 0 e 1.

Definindo Funções

Sintaxe:

```
function nomeDaFuncao(parametros) {  
    // operações da função  
    return resultado;  
}
```

Exemplo:

```
...  
function max(a, b) {  
    if (a > b)  
        return a;  
    else  
        return b;  
}  
...  
alert('Maior nro:' + max(5,8));
```

OBS: Funções em **JavaScript** não precisam necessariamente retornar um valor. Caso a declaração **'return'** não seja utilizada, o valor ***undefined*** será automaticamente retornado.

Definindo Funções – Argumentos Opcionais

Caso um argumento deixe de ser fornecido ao chamar a função, o parâmetro correspondente receberá o valor **undefined**;

Exemplo:

```
function potencia(base, expoente) {  
    if (expoente == undefined)  
        expoente = 2;  
  
    var resultado = 1;  
    for (var i = 0; i < expoente; i++)  
        resultado = resultado * base;  
    return resultado;  
}  
  
console.log(potencia(2,5)); // a saída será 32  
console.log(potencia(3));  // a saída será 9 (o exp. padrao é 2)
```

Funções e Escopo de Variáveis

- Variáveis definidas dentro de funções têm **escopo local** e podem ser acessadas apenas dentro delas;
- Variáveis definidas fora das funções têm **escopo global** e podem ser acessadas **por qualquer script ou função da página Web**;
- Toda variável global definida no código JavaScript pode ser acessada como uma propriedade do objeto **window**:

window.nomeDaVariavel;

Funções e Escopo de Variáveis

Exemplo:

```
<!DOCTYPE html>
<html>
<head><title>Escopo de Variaveis</title></head>
<body>

<script>

var opcaoAtiva = 1; // Esta é uma variável global

function avancaOpcao() {
    opcaoAtiva++;
}

function retornaOpcao() {
    opcaoAtiva--;
}

avancaOpcao();
avancaOpcao();
retornaOpcao();

</script>

<body>

    <input type="button" onclick="alert('Valor da var. global: ' + window.opcaoAtiva)"
value="Clique aqui!">

</body>
</html>
```

Funções - Exemplo

O exemplo a seguir define duas funções: **fatorial** e **testeFatorial**

A função **testeFatorial** é chamada quando o usuário clica no botão. Ela solicita um número, chama a função *fatorial* para realizar o cálculo e apresenta o resultado.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Definindo Funções em JavaScript - Exemplo</title>
    <script>

      function fatorial(n) {
        var total = 1;
        for (var i = 2; i <= n; i++) {
          total = total * i;
        }
        return total;
      }

      function testeFatorial() {
        var num = prompt("Informe um numero inteiro positivo: ");
        var numInt = parseInt(num); // converte a string em inteiro
        var fat = fatorial(numInt);
        document.write('O fatorial eh: ' + fat);
      }

    </script>
  </head>
  <body>
    <input type="button" onclick="testeFatorial()" value="Clique para calc. o fatorial">
  </body>
</html>
```


Funções Definidas em Arquivo Externo - Exercício

1. Altere o exemplo anterior colocando as funções em um arquivo externo de nome **script.js**

```
/* arquivo script.js */  
function fatorial(n) {  
    var total = 1;  
    for (var i = 1; i <= n; i++) {  
        total = total * i;  
    }  
    return total;  
}  
  
function testeFatorial() {  
    var num = prompt("Informe um numero inteiro positivo: ");  
    var numInt = parseInt(num); // converte a string em inteiro  
    var fat = fatorial(numInt);  
    document.write('O fatorial eh: ' + fat);  
}
```

2. Coloque uma referência para o script no arquivo HTML e teste a página.

```
<!DOCTYPE html>  
<html>  
    <head><title>Cabeçalhos</title>  
    <script src="script.js"></script></head>  
    <body>  
        <input type="button" onclick="testeFatorial()" value="Clique para calcular">  
    </body>  
</html>
```

Depurando Código JavaScript

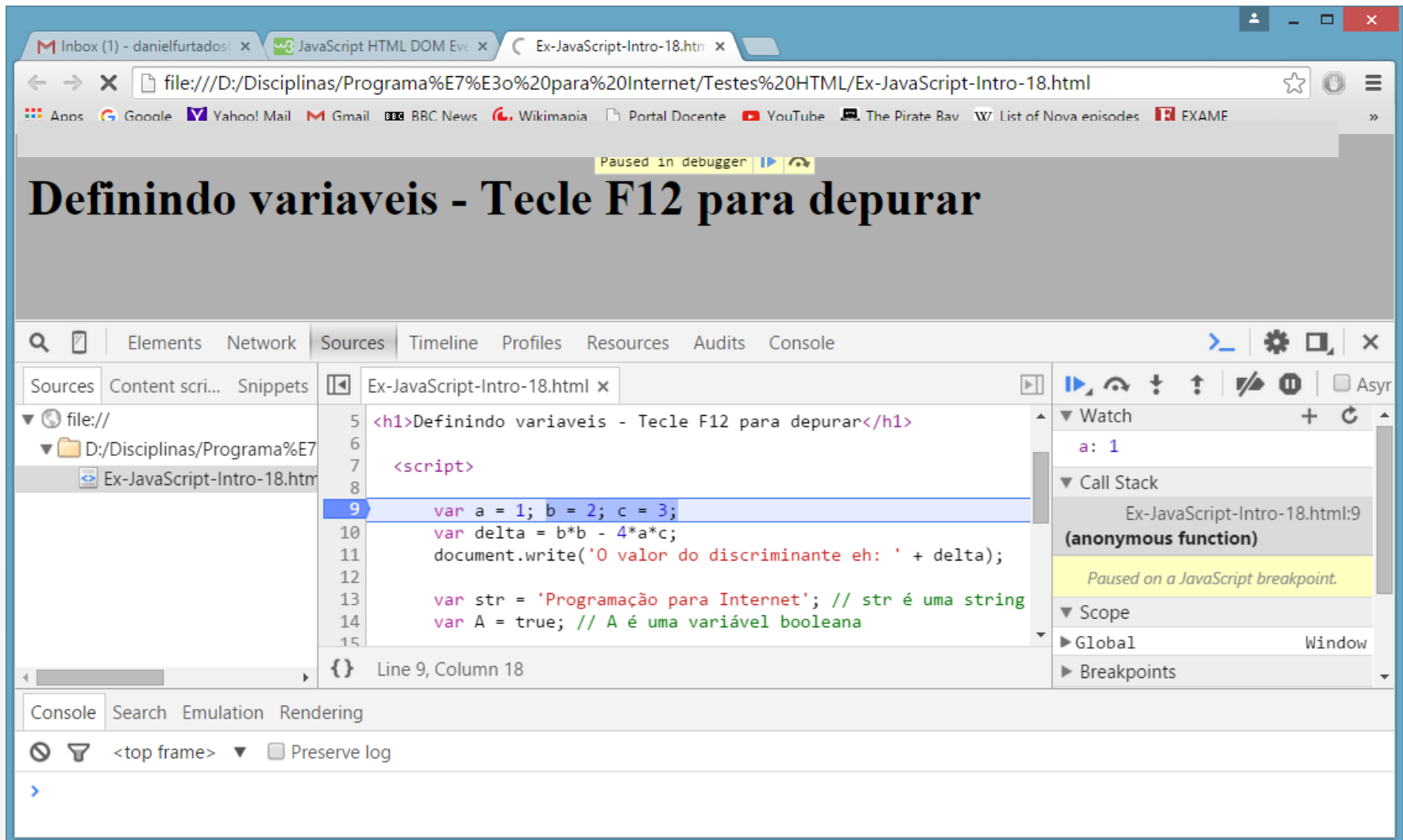
Depurando Código JavaScript

A depuração do código JavaScript pode ser realizada nos principais navegadores;

No caso do Google Chrome, siga os passos a seguir:

- Pressione a tecla F12;
- Escolha a aba *sources* e selecione o arquivo contendo o script;
- Clique sobre o número da linha de código que deseja interromper a execução;
- Recarregue a página (F5);
- Tecle F10 para executar o script passo a passo ou F8 para continuar a execução até o próximo *breakpoint*;
- Acompanhe o valor das variáveis no painel lateral (*watch* → “+” → *digitar nome da variável*)

Depurando Código JavaScript



Depurando Código JavaScript

- Em algumas situações pode ser necessário depurar um código JavaScript que foi carregado dinamicamente pela página;
- Nessas situações pode ser útil escrever o comando ***debugger*** no código onde se deseja pausar a execução do script. Neste caso, a execução será automaticamente interrompida naquela linha do código quando o navegador estiver no modo de depuração.
 - ***Dica:*** insira o comando ***debugger*** em algum trecho do código anterior e recarregue a página no modo de depuração do navegador.

Exercício 1

1. Crie uma página HTML contendo um título e um parágrafo;
2. Crie uma função JavaScript para alterar a cor do título para azul quando o usuário clicar sobre o mesmo;
3. Crie uma função JavaScript para colocar o texto do parágrafo em negrito quando o usuário passar o ponteiro do mouse sobre o mesmo;
4. Crie uma função JavaScript para voltar o texto do parágrafo para *normal* (sem negrito) quando o usuário afastar o ponteiro do mouse do parágrafo;
5. Crie uma função JavaScript para trocar todo o texto do parágrafo pela mensagem “Obrigado pelo click!” quando o usuário clicar sobre o mesmo;
6. Execute o código JavaScript passo a passo utilizando o modo de depuração (F12)

OBS: não utilize a biblioteca jQuery

Exercício 2

1. Crie uma página HTML que contenha uma imagem qualquer apresentada no seu tamanho natural (sem ajuste de largura e altura);
2. Crie uma função JavaScript para dobrar o tamanho da imagem quando o usuário colocar o ponteiro do mouse sobre a mesma;
3. Crie uma função JavaScript para voltar a imagem ao seu tamanho original quando o usuário afastar o ponteiro do mouse da mesma;

Dicas

- Utilize as propriedades *naturalWidth* e *naturalHeight* do objeto da imagem no código JavaScript para ter acesso ao tamanho original da imagem;
- Altere o tamanho da imagem no código JavaScript utilizando as propriedades CSS *width* e *height*;
- Ao definir um novo valor para as propriedades *width* e *height*, concatene os valores calculados com a string “px”

OBS: não utilize a biblioteca jQuery

Exercício 3

1. Crie uma cópia do arquivo do exercício anterior
2. Remova as funções de aumento/retorno do tamanho da imagem;
3. Crie uma função JavaScript para colocar uma borda azul de 5 pixels de espessura na imagem quando o usuário colocar o ponteiro do mouse sobre a mesma;
4. Crie outra função JavaScript para retirar a borda quando o usuário afastar o ponteiro do mouse da imagem;
5. Após implementar e testar as funções anteriores, insira 5 cópias da imagem no arquivo HTML. Em seguida, faça as devidas adaptações no código para que a borda seja mostrada apenas na imagem apontada pelo mouse. Não utilize o método *getElementById*. Peça ajuda ao professor.

Exercício 4

Faça uma página em HTML com JavaScript que solicite ao usuário os valores dos coeficientes de uma equação do segundo grau e calcule e apresente o seu discriminante (“delta”);

- Crie uma função de nome `CalculaDiscriminante` que receba três números (parâmetros) e devolva o valor do respectivo discriminante ($\text{discriminante} = b^2 - 4ac$)
- Crie uma função de nome `TesteDiscriminante` para solicitar os coeficientes ao usuário (utilizando o método `prompt`), fazer a chamada da função `CalculaDiscriminante` e apresentar o resultado;
- Crie um botão “Calcular Discriminante” para fazer a chamada à função `TesteDiscriminante`;

Observações:

- As funções JavaScript devem ser inseridas adequadamente dentro da seção `head` do documento HTML;
- Execute o script normalmente;
- Execute o script passo a passo utilizando o modo de depuração.

Vetores, *Strings* e Objetos

Vetores

- Em JavaScript, **vetores** (*arrays*) podem ser definidos colocando-se os elementos entre colchetes (separados por vírgula);
- O primeiro elemento do *array* possui índice 0;
- Os vetores são tratados como objetos. Por exemplo, o número de elementos pode ser resgatado por meio da propriedade *length*;
- Exemplo:

```
<script>

    var vetorDeNumeros = [4,1,6,3,8,1];
    var soma = 0;
    var n = vetorDeNumeros.length;
    for (var i=0; i < n; i++)
        soma = soma + vetorDeNumeros[i];

    alert('A soma dos elementos do vetor eh: ' + soma);

</script>
```

Vetores

É possível ter vetores com elementos de tipos diferentes;

Exemplo:

```
<script>  
  
    var vet = [5, 'a', 3, 'java', 2];  
  
    console.log(vet[1]); // a saída será o caractere 'a'  
    console.log(vet[2])  // a saída será o número 3  
  
</script>
```

Vetores

Em JavaScript, os vetores também podem ser utilizados como uma estrutura de dados *pilha*:

- O método *push* insere um novo elemento no final ('topo') do vetor;
- O método *pop* remove e retorna o último elemento do vetor;

Exemplo:

```
<script>

    var vet = [];

    vet.push("Programacao");
    vet.push("para");
    vet.push("Internet");
    console.log(vet); // A saída será: ["Programacao", "para", "Internet"]

    var ultimo = vet.pop();
    console.log(ultimo); // A saída será: 'Internet'
    console.log(vet);    // A saída será: ["Programacao", "para"]

</script>
```

Strings

- Em JavaScript, *strings* podem ser definidas utilizando aspas simples ou duplas;
- Para acessar um caracter da string, utilize o método *charAt*

```
var str = "Esta eh uma string";  
var x = str.charAt(0); // x receberá 'E';
```

- Carecteres especiais podem ser adicionados às *strings* por meio do caracter de escape \

```
var x = 'It\'s alright';
```

- Outras propriedades e métodos para manipulação de strings

Propriedade/Método	Descrição
length	Retorna o comprimento (número de caracteres) da string
indexOf	Retorna a posição (índice) da primeira ocorrência de um texto na string (ou -1 caso não encontrado)
lastIndexOf	Retorna a posição (índice) da última ocorrência de um texto na string
substr(s,n)	Retorna uma substring de n caracteres começando na posição s
split	Divide a string em várias partes de acordo com um separador

Objetos Simples em JavaScript

- Objetos simples em JavaScript (*PlainObject*) podem ser utilizados para armazenar uma coleção de propriedades de entidades do mundo real;
- Tais objetos podem ser definidos por meio de uma lista de pares do tipo **nomeDaPropriedade** : **valor**, separados por vírgula e colocados entre **colchetes**. Exemplo:

```
var pessoa = {  
  pNome: "John",  
  uNome: "Doe",  
  idade: 50,  
  eyeColor: "blue"  
};  
  
alert(pessoa.pNome); // acessa a propriedade pNome do objeto pessoa  
alert(pessoa.idade); // acessa a propriedade idade do objeto pessoa
```

Ref: adaptado de [w3schools.com](https://www.w3schools.com)

Formas de Acessar as Propriedades dos Objetos

As propriedades dos objetos JavaScript podem ser acessadas de três formas:

1. *nomeObjeto.propriedade* // *pessoa.idade*
2. *nomeObjeto["propriedade"]* // *pessoa["idade"]*
3. *nomeObjeto[expressão]* // *x = "idade"; pessoa[x]*

Acessando as Propriedades dos Objetos - Exemplo

```
<!DOCTYPE html>
<html>
<body>

<script>

var carro = { marca : "BMW", modelo : "X6", cor : "Branco" };

document.write( carro.marca );
document.write( carro.modelo );
document.write( carro.cor );

document.write( carro['marca'] );
document.write( carro['modelo'] );
document.write( carro['cor'] );

x = 'marca';
document.write(carro[x]);

</script>

</body>
</html>
```

Document Object Model (DOM)

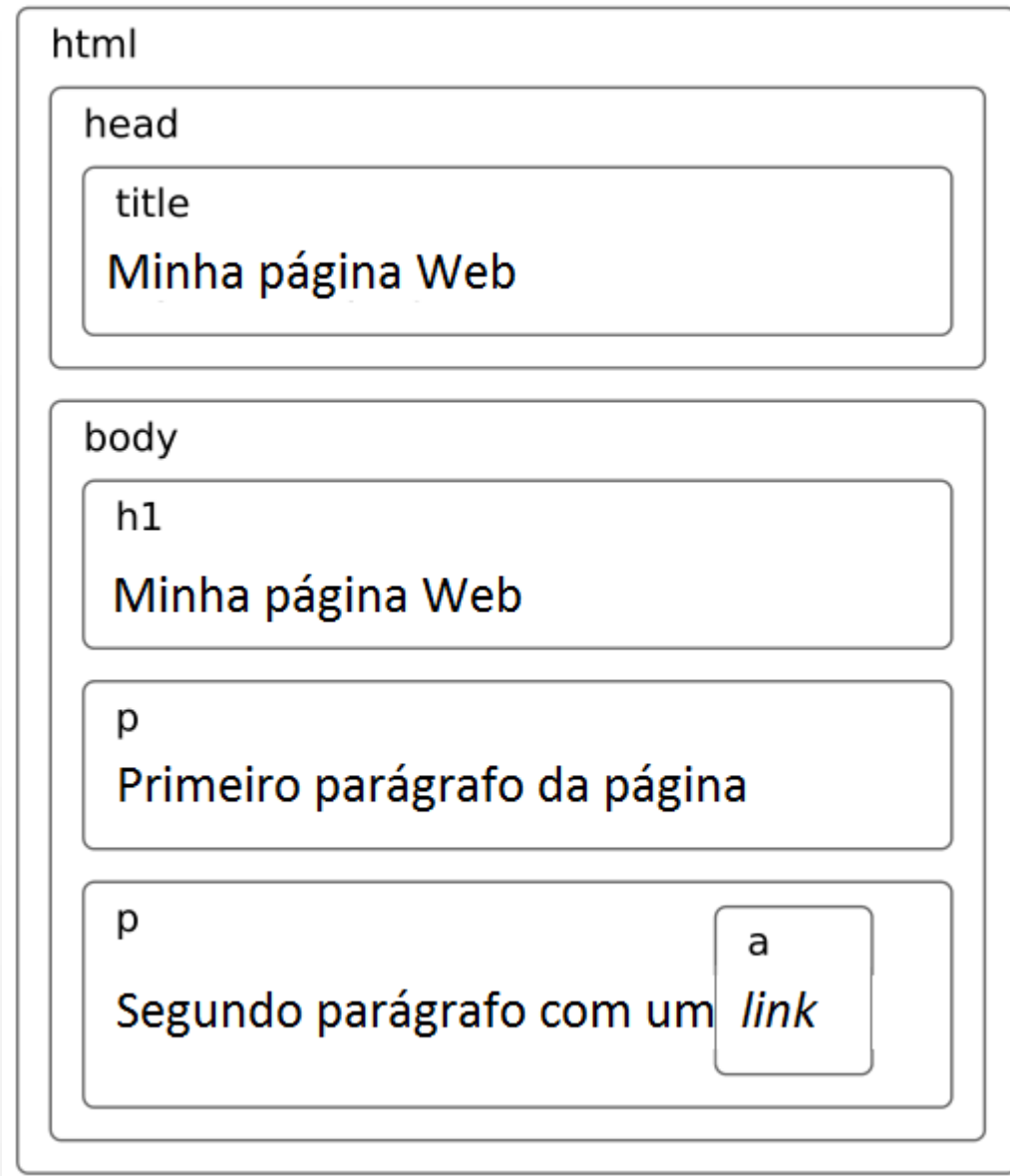
Document Object Model (DOM)

- DOM: *Document Object Model*
- É um modelo utilizado pelos navegadores de Internet no qual um documento HTML é representado como uma coleção hierárquica de objetos;
- Em outras palavras, DOM é uma representação em estrutura de árvore de todos os elementos de uma página Web;

Document Object Model (DOM)

```
<!DOCTYPE html>
<html>
<head>
<title>Minha página Web</title>
</head>
<body>

<h1>Minha página Web</h1>
<p>Primeiro parágrafo da página</p>
<p>Segundo parágrafo com um
<a href="http://www.ufu.br">link</a>.
</p>
</body>
</html>
```



Document Object Model (DOM)

Assim, tudo no documento HTML corresponde a um nó:

- O documento propriamente dito é um nó (do tipo *document*);
- Todos os elementos HTML, como <p>, <a>, <h1>, etc., são nós (do tipo *element*);
- Todos os atributos HTML são nós (do tipo *attribute*)
- O conteúdo dos elementos HTML são nós (do tipo *text*)
- E até mesmo os comentários são nós (do tipo *comment*)

Document Object Model (DOM)

■ Alguns métodos para manipulação da estrutura DOM

document.createElement
document.createTextNode
node.childNodes

node.appendChild
node.removeChild
node.hasChildNodes
node.cloneNode

node.firstChild
node.lastChild
node.parentNode

■ Exemplos:

- <https://www.impressivewebs.com/10-essential-dom-methods-techniques-for-practical-javascript/>
- https://www.w3schools.com/jsref/met_node_appendchild.asp

Document Object Model (DOM)

- A estrutura de objetos do documento pode ser acessada por meio do objeto *document*, que é o nó raiz da hierarquia;
- Algumas formas de resgatar elementos da estrutura DOM:
 - **document.getElementById**("id_do_elemento"): retorna o elemento HTML que possui o atributo **id** igual a "*id_do_elemento*";
 - **document.getElementsByName**("nome_do_elemento"): retorna a coleção de elementos que possuem o atributo **name** igual a "*nome_do_elemento*";
 - **document.getElementsByTagName**("tag_name"): retorna a coleção de elementos cujo nome da tag é "*tag_name*";
 - **document.getElementsByClassName**("class_name"): retorna a coleção de elementos que utilizam a class CSS "*class_name*";

DOM – *getElementById* – Exemplo 1a

```
<!DOCTYPE html>
<html>
<head><title>Teste JavaScript</title></head>
<body>

Base: <input type="text" id="base">
Expoente: <input type="text" id="expoente">

<button onClick="calcularPotencia()">Calcular!</button>
<p id="pResultado">O resultado aparecera neste paragrafo!</p>

<script>
  function calcularPotencia() {

    var inputBase = document.getElementById("base");
    var inputExpo = document.getElementById("expoente");

    var base = parseInt(inputBase.value);
    var expo = parseInt(inputExpo.value);

    var res = 1;
    for (var i=0; i < expo; i++) {
      res = res * base;
    }
    document.getElementById("pResultado").innerHTML = res;
  }
</script>
</body>
</html>
```


DOM – *getElementsByTagName* – Exemplo 1b

```
<!DOCTYPE html>
<html>
<head><title>Teste JavaScript</title></head>
<body>
```

Base: <input type="text" id="base">

Expoente: <input type="text" id="expoente">

<button onClick="calcularPotencia()">Calcular!</button>

<p id="pResultado">O resultado aparecera neste paragrafo!</p>

```
function calcularPotencia() {
```

```
    // getElementByTagName retorna um vetor contendo todos os elementos
    // do tipo 'input'. O índice entre colchetes acessa um elemento específico.
```

```
    var inputBase = document.getElementsByTagName("input")[0];
```

```
    var inputExpo = document.getElementsByTagName("input")[1];
```

```
    var base = parseInt(inputBase.value);
```

```
    var expo = parseInt(inputExpo.value);
```

```
    var res = 1;
```

```
    for (var i=0; i < expo; i++) {
        res = res * base;
```

```
    }
```

```
    document.getElementById("pResultado").innerHTML = res;
```

```
}
```

```
</script>
```

```
</body></html>
```

DOM – *getElementsByTagName* – Exemplo 1b

```
<!DOCTYPE html>
<html>
<head><title>Teste JavaScript</title></head>
<body>

Base: <input type="number" size="10" name="base">
Expoente: <input type="number" size="10" name="expo">

<button onClick="calcularPotencia()">Calcular!</button>
<p id="pResultado">O resultado aparecera neste paragrafo!</p>

<script>
  function calcularPotencia() {

    // getElementsByTagName retorna um vetor contendo todos os elementos
    // do tipo 'input'. Aqui, um elemento em particular é acessado utilizando o
    // valor de atributo 'name' (base e expo)

    var inputBase = document.getElementsByTagName("input")["base"];
    var inputExpo = document.getElementsByTagName("input")["expo"];

    var base = parseInt(inputBase.value);
    var expo = parseInt(inputExpo.value);

    var res = 1;
    for (var i=0; i < expo; i++) {
      res = res * base;
    }
    document.getElementById("pResultado").innerHTML = res;
  }
</script>
</body>
</html>
```

DOM - Alterando Estilos CSS

- Propriedades CSS também podem ser exploradas utilizando a estrutura DOM e JavaScript;
- Basta resgatar o objeto desejado, utilizar o campo **style** e o nome da propriedade CSS sem o hífen;
- Valores de medidas com unidade, como 10px, devem ser fornecidos entre aspas

Exemplo:

```
<!DOCTYPE html>
<html><head><title>CSS e JavaScript</title></head>
  <body>
    <div id="divTexto" onClick="colorir()">Clique aqui para mudar o estilo!</div>

    <script>
      function colorir() {
        var elementoDiv = document.getElementById("divTexto");
        elementoDiv.style.backgroundColor = '#AA0000';
        elementoDiv.style.color = 'white';
        elementoDiv.style.textAlign = 'center';
        elementoDiv.style.height = '20px';
      }
    </script>
  </body></html>
```

DOM – Alterando Estilos CSS – Exemplo

```
<!DOCTYPE html>
<html>
<body>

<h1 onmouseover="mOver(this)" onmouseout="mOut(this)">Passe o mouse aqui</h1>

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
    style="background-color:red; width:140px; height:20px; padding:40px;">
    Passe o mouse aqui</div>

<script>
function mOver(obj) {
    obj.innerHTML = "Obrigado! :-)";
    obj.style.backgroundColor = 'lightgreen';
}

function mOut(obj) {
    obj.innerHTML = "Passe o mouse aqui";
    obj.style.backgroundColor = 'red';
}
</script>

</body>
</html>
```

Collections do Objeto *document*

- Alguns elementos do documento HTML (como formulários, links e imagens) também podem ser acessados por meio de propriedades especiais denominadas *collections*:
 - `document.forms`: retorna uma coleção com todos os formulários da página;
 - `document.images`: retorna uma coleção com todas as imagens () da página;
 - `document.anchors`: retorna uma coleção com todos links (<a>) da página;
- Exemplos:

```
// acessa o 1º formulário da coleção de formulários da página  
var form1 = document.forms[0]
```

```
// acessa o formulário de nome 'form1' da coleção de  
// formulários da página  
var form1 = document.forms["form1"]
```

Validação de Formulários

- JavaScript pode ser utilizada para validar formulários no lado *cliente*, pelo navegador;
- Pode-se utilizar o atributo *onSubmit* do elemento **<form>** em conjunto com uma função JavaScript para realizar a validação;
- Neste caso, a função deve verificar o conteúdo dos campos e retornar **true** quando todos os campos forem válidos ou **false** quando algum campo for inválido;
- O formulário será submetido apenas quando o valor de retorno da função for verdadeiro;

Validação de Formulários - Exemplo

```
<!DOCTYPE html>
<html>
<head><title>Validando Formulários com JavaScript</title>
<script>

function validaForm()
{
    var usuario = document.forms["form1"]["usuario"].value;
    if (usuario == null || usuario == "") {
        alert("O campo usuario deve ser preenchido");
        return false;
    }
    var senha = document.forms["form1"]["senha"].value;
    if (senha == null || senha == "") {
        alert("O campo senha deve ser preenchido");
        return false;
    }

    return true;
}

</script>
</head>

<body>
    <form name="form1" action="login.php" onSubmit="return validaForm()" method="post">
        Usuário: <input type="text" name="usuario">
        Senha: <input type="password" name="senha">
        <input type="submit" value="Enviar">
    </form>
</body></html>
```

Atributos de Restrição da HTML

- A linguagem HTML disponibiliza uma série de atributos para impor restrições no preenchimento de campos de formulário do tipo **<input>**;
- Usando tais atributos, a restrição é naturalmente implementada pelo navegador.

Atributo	Descrição
<code>disabled</code>	Indica que o elemento input está desabilitado para edição
<code>max</code>	Especifica o valor máximo para o elemento input
<code>min</code>	Especifica o valor mínimo permitido para o elemento input
<code>pattern</code>	Especifica uma expressão regular que define os valores permitidos
<code>required</code>	Indica que o campo é de preenchimento obrigatório

Atributos de Restrição da HTML

Exemplo

```
<!DOCTYPE html>
<html>
<head><title>Restringindo Campos de Formulários com Atributos
HTML</title>

<form name="form1" action="login.php" method="post">

Código: <input type="codigo" name="codigo" value="100" disabled>
Usuário: <input type="text" name="usuario" required>
Senha: <input type="password" name="senha" required>
Email: <input type="email" name="email" required>
Idade: <input type="number" name="idade" min="18" max="60" required>
Estado: <input type="text" name="estado" pattern="[A-Za-z]{2}">

<input type="submit" value="Enviar">

</form>
</body>
</html>
```

Exercício 5

- Altere a solução do exercício 4 para que os coeficientes da equação do 2º grau sejam solicitados ao usuário em campos de formulário do tipo `<input>`;
 - Utilize código JavaScript para acessar o conteúdo dos campos;
 - O resultado deve ser apresentado ao usuário em um elemento HTML do tipo `<H1>`;

Exercício 6

- Elabore uma página HTML para calcular o peso ideal de uma pessoa. A página deverá exibir um formulário com dois campos:
 - Um campo do tipo *radio* para informação do sexo da pessoa;
 - Um campo de texto convencional para entrada da altura da pessoa;
- A página deverá calcular e apresentar o peso ideal utilizando funções JavaScript. Utilize as fórmulas a seguir (onde *h* é a altura, em metros):
 - Para homens: $\text{peso ideal} = (72.7 * h) - 58$
 - Para mulheres: $\text{peso ideal} = (62.1 * h) - 44.7$

Dica: para identificar, no código JavaScript, qual foi o sexo escolhido pelo usuário, utilize como base o código a seguir:

```
var radios = document.getElementsByName("sexo");
for (var i = 0; i < radios.length; i++)
{
    if (radios[i].checked)
    {
        // realizar alguma operacao com radios[i].value
    }
}
```

Referências

- www.eloquentjavascript.net
- www.w3schools.com/js/