Fai Faculdades - Itapiranga/SC
Gestão da Tecnologia da Informação
Professor: André Klunk **E-mail: andre.gti@seifai.edu.br**

PARTE 1

JAVASCRIPT

1. Introdução ao JavaScript

JavaScript é um recurso (nem todos os desenvolvedores web o considerem uma linguagem) baseada em scripts, usada para enriquecer a ação do usuário e acrescentar recursos dinâmicos à interface

Desenvolvida pela Netscape para funcionar entre plataformas, atua embutida em programas como os browsers web e permite a conexão entre os elementos das páginas e o controle sobre eles

JavaScript pode interagir com diversos componentes em HTML e CSS das páginas web para realizar mudanças em tempo real ou para responder a ações dos usuários

Apesar do nome, não tem relação com a linguagem Java, desenvolvida pela Sun MicroSystems

Baseada em objetos, utiliza-os para o reconhecimento de suas estruturas de dados. Do mesmo modo que o mundo que nos cerca é composto de objetos, a Javascript reconhece unidades de informação ou unidades funcionais como objetos, sendo que cada uma delas tem propriedades que as identificam, como acontece com os objetos concretos

É uma linguagem interpretada, as instruções não precisam ser compiladas para funcionar, pois são divididas em pequenos componentes, analisados um a um pelo browser de cada usuário. Diversos conjuntos de instruções se somam para gerar uma determinada funcionalidade

As ações do usuário se efetivam através destas instruções, enviadas para o browser, os scripts. Na maioria dos casos, os scripts são executados na máquina do usuário, não exigem resposta do servidor. No entanto, a linguagem também roda através de scripts localizados em servidores web para gerar arquivos em HTML

Pode-se criar com Javascript um texto animado na base da página, um rollover de botão, um relógio que atualiza a hora de quem acessa a página, calendários, menus, efeitos de fundo, jogos, entre outros recursos.

Os scripts desenvolvidos em Javascript podem ficar localizados dentro do código HTML da página web, podem ser copiados e colados de uma página para outra e ter seu conteúdo e funcionalidade adaptados

A linguagem apresenta três versões:

- JavaScript: baseada na linguagem criada pela Netscape, foi aplicada pela primeira vez no Netscape 2.0. É a versão mais popular
- JScript: versão criada pela Microsoft para competir com a linguagem criada pela Netscape
- ECMAScript: Standard internacional criado em 1996 para a linguagem, que cobre todo o seu escopo

Enquanto o Internet Explorer reconhece JScript e JavaScript, o Netscape só reconhece o segundo. Como as marcações são praticamente as mesmas, é mais fácil considerar que os scripts escritos em JavaScript rodam em todos os browsers

O W3C publicou uma recomendação (DOM) segundo a qual Javascript vai continuar em uso em XHTML, pois é uma linguagem de uso praticamente universal, roda em qualquer plataforma e na maioria das versões de browsers, mesmo os mais antigos.

De qualquer forma, para garantir a correta interpretação dos scripts, os desenvolvedores precisam se adaptar ao uso de código compatível com os standards web

Apesar do uso disseminado, seu uso não é tão disseminado quanto afirmam os colaboradores do W3C. Certa de 10% dos usuários (de acordo com TheCounter.com - pesquisa em público internacional) não usam ou não habilitam o uso de Javascript em seus browsers. Por isto, a interface não deve depender da habilitação deste recurso para funcionar JavaScript

Os principais browsers do mercado, Firefox, Mozilla, Opera, Netscape, Safari, Internet Explorer e outros mais, suportam a linguagem JavaScript. Ela é utilizada para a validação de formulários, ler e inserir conteúdo em uma página, criar cookies, menus dinâmicos e diversos outros recursos.

1.2. Inserindo o JavaScript em uma página HTML

Para inserirmos o JavaScript em uma página HTML, fazemos uso da tag <script> e do atributo type com o valor: text/javascript. O que resultaria no seguinte:

```
<script type="text/javascript">
//códigos JavaScript
</script>
```

Ao iniciarmos a tag <script>, dizemos o tipo de script utilizado type="text/javascript" e assim o browser entenderá que no conteúdo dessa tag há comandos JavaScript para serem executados e os reconhecerão. Para finalizar a execução dos códigos usa-se o final da tag como </script>.

Há três formas diferentes de inserirmos o JavaScript em uma página HTML, são elas: um arquivo externo -.js, no head, no body ou utilizando os três de uma só vez.

1.2.1 Arquivo (script) externo

O arquivo externo é chamado desta forma:

. . .

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

Ou seja, através do atributo src da tag <script>, indicamos a localização do arquivo externo, para que assim o browser possa localizá-lo e executá-lo.

Em arquivos externos não utilizamos a tag <script>, somente os códigos.

As vantagens de se trabalhar dessa forma são:

- Facilidade na manutenção: Uma vez que o script está localizado em apenas um arquivo, facilita a edição ou correção dos códigos.
- Carregamento mais rápido da página: O arquivo externo é armazenado no cache do navegador. Assim, evita-se carregá-lo toda vez que a página for chamada.
- Semântico: O arquivo externo separa a camada de comportamento (JavaScript) da camada de conteúdo (HTML).

1.2.2 Script no head da página

O script é inserido desta forma:

```
...
<head>
<script type="text/javascript">
//códigos JavaScript
</script>
</head>
...
```

Os códigos JavaScript localizado no head da página são executados ao serem chamados, ou seja, quando algum evento for provocado, como o evento onclick, onmouseover, onload, dentre outros.

Nessa forma o script é carregando antes de alguém utilizá-lo, ou seja, antes do carregamento do conteúdo do body.

1.2.3 Script no body da página

O script é inserido desta forma:

```
...
<body>
<script type="text/javascript">
//códigos JavaScript
</script>
</body>
...
```

Os códigos inseridos no body da página são inicialmente executados enquanto a página é carregada mas também podem ser chamados quando algum evento for provocado.

1.3. A tag <noscript>

A tag <noscript> nos permite disponibilizar um conteúdo alternativo ao disposto via script. Isso significa dizer que, se o usuário não tiver suporte ao JavaScript em seu browser, ele verá o conteúdo inserido dentro dela e, caso tenha suporte, o seu conteúdo é omitido.

Exemplo:

```
<script type="text/javascript">
alert("Parabéns, você está aprendendo JavaScript.");
</script>
<noscript>
<p>Parabéns, você está aprendendo JavaScript.</p>
</noscript>
```

No exemplo acima será exibida a mensagem, "Você está aprendendo JavaScript", através do alert() - uma função do JavaScript que exibe uma mensagem através de um popup box - para quem tem suporte ao JavaScript. Caso contrário, mesma mensagem aparecerá para quem não tem suporte ao JavaScript, porém através da tag <noscript> na forma de texto.

A tag <noscript> é muito importante quando se trata de acessibilidade, pois oferece um conteúdo alternativo para os usuários sem suporte à linguagem em questão.

1.4. Comentários em JavaScript

Os comentários na linguagem JavaScript tem função similar às demais linguagens, ou seja, de explicar o que determinado comando faz, nota de algum autor, lembretes, dentre outras.

Tudo que estiver englobado pelos comentários é ignorado pelo interpretador do JavaScript.

Há dois tipos de comentários, são eles: // e /* */. O primeiro é o comentário de uma linha somente, enquanto o segundo suporta várias.

Exemplos:

```
<script type="text/javascript">
// Isto é um comentário de uma linha.
/* Isto é um comentário de várias linhas, e o interpretador ignora todo esse conteúdo. O que nos permite a criação de notas e lembretes em nossos códigos. */
</script>
```

Um detalhe importante que não podemos esquecer, é a utilização do comentário em HTML ao utilizar códigos JavaScript in-line (disposto no head ou body). Os browsers que não suportam o JavaScript irão exibi-lo como se fosse parte do conteúdo da página. Para prevenir essa prática, utilizamos o comentário em HTML.

Exemplo:

```
...
<head>
<script type="text/javascript">
<!--
//códigos JavaScript
//-->
</script>
</head>
...
```

Porém, você pode se perguntar: Para que serve as duas barras (//) antes do fechamento do comentário? Bom, elas previnem que o interpretador do JavaScript as interpretem.

1.5 "Case sensitive"

JavaScript é "case sensitive", o que significa dizer, que o interpretador diferencia minúsculas de maiúsculas.

1.6 Ponto e vírgula (;) no final das declarações

Em JavaScript a utilização do ; (ponto e vírgula) é quase que opcional. Digo quase, porque se você dispor de duas declarações de código na mesma linha, precisará do ; (ponto e vírgula).

Exemplo:

```
alert("Estes detalhes..."); alert("... são importantes");
alert("Notou...")
alert("A diferença?")
```

1.7 Variáveis

Variáveis permitem identificar valores (valores primitivos, arrays ou objectos) através de um nome. O valor diz-se o conteúdo da variável.

Vejamos um exemplo de como declarar uma variável:

```
var nomeVariavel = "valorVariavel";
ou
nomeVariavel = "valorVariavel";
```

Você pode criar uma variável com ou sem a declaração var, mas é aconselhável utilizá-la.

Quando criamos uma variável dentro de uma função, a tornamos local, ou seja, acessível somente dentro da função em que foi criada. Podemos até criar outras variáveis com o mesmo nome e com valores diferentes, porém elas terão validade somente dentro da função em que foi declarada.

Já as variáveis globais, são declaradas fora das funções e ficam acessíveis a toda e qualquer função na página em que a variável esteja.

1.8 Operadores

Operadores são símbolos utilizados para atuar sobre valores. Conheceremos os diferentes operadores utilizados no JavaScript.

Operadores aritméticos

Operador	Descrição	Exemplo	Resultado
+	Adição	x=2 y=8 x+y	10
-	Subtração	x=10 y=3 x-y	7
*	Multiplicação	x=4 x*5	20
/	Divisão	15/5	3
%	Módulo (restante da divisão)	5%2 10%8 10%2	1 2 0
++	Incrementos	x=5 x++	x=6
--	Decrementos	x=5 x--	x=4

Operadores de comparação

Operador	Descrição	Exemplo	Resultado
==	é igual a	5==8	falso
===	é igual a (compara o valor e o tipo)	x=5 e y="5" x==5 x===5	verdadeiro falso
!=	não é igual	5!=8	verdadeiro
>	é maior que	5>8	falso
<	é menor que	5<8	verdadeiro
>=	é maior que ou igual a	5>=8	falso
<	é menor que ou igual a	5<=8	verdadeiro

Operadores lógicos

Operador	Descrição	Exemplo	Resultado
&&	e	x=6	verdadeiro

Operador	Descrição	Exemplo	Resultado
		y=3 (x < 10 && y > 1)	
||	ou	x=6 y=3 (x==5 || y==5)	falso
!	não	x=6 y=3 !(x==y)	verdadeiro

Operadores de atribuição

Operador	Exemplo	É o mesmo que
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=Y	x=x%y

Operador condicional

Operador condicional é aquele que atribui um valor a uma determinada variável com base em uma condição, ou seja, se a condição for verdadeira, teremos um valor x, caso contrário um valor y.

Sintaxe desse tipo de condição:

```
nomeDaVariável = (condição) ? valorSeVerdadeiro : valorSeFalso;
```

Exemplo:

```
var fruta = "Maçã";

var resposta = (fruta == "Maçã") ? "A fruta é a Maçã." : "Não sei qual é a fruta.";
```

No exemplo acima fizemos o seguinte processo: inicialmente temos uma variável denominada fruta com o valor Maçã; em seguida, a variável resposta com um valor condicional, ou seja, se o valor da variável fruta for igual a Maçã, teremos o valor **A fruta é a Maçã**, caso contrário, teremos **Não sei qual é a fruta**.

1.9 Condições

Durante o desenvolvimento dos nossos scripts, sentimos a necessidade de executar determinadas ações baseados em uma determinada condição, ou seja, se uma condição for verdadeira (true) será executada a ação X, caso contrário, se for falsa (false), a ação Y.

Em JavaScript temos as seguintes declarações condicionais:

1.9.1 Declaração if

Utilizamos essa declaração quando necessitamos que somente uma parte do código seja executada e quando uma determinada condição for verdadeira.

```
if (condição) {
 //código a ser executado quando a condição for verdadeira.
}
else {
 //código a ser executado quando a condição for falsa.
}
```

1.9.2 Declaração switch

Utilizamos também essa declaração quando desejamos selecionar um bloco de código entre vários outros. É uma forma mais prática para alguns casos.

```
switch(expressão) {
 case valor 1:
        //código a ser executado se a expressão = valor 1;
        Break
 case valor 2:
         //código a ser executado se a expressão = valor 2;
         Break
  default:
         //código a ser executado se a expressão for diferente do
       valor 1 e valor 2;
}
```

Algumas considerações a cerca do switch.

- A expressão na maioria das vezes será uma variável;
- O valor da expressão é comparado com os valores de cada caso (case) da estrutura. Caso ocorra um valor semelhante, o código a ele associado é executado;
- O break previne que o próximo caso (case) seja executado automaticamente;
- O valor padrão (default) é utilizado caso nenhum dos valores dos cases coincidam com o valor da expressão.

2. Popup Boxes

Os popup boxes no JavaScript nos permitem alertar o usuário sobre algum acontecimento, pedir-lhe uma confirmação para que uma determinada ação possa ser executada e pedir-lhe que nos informe algum dado.

Assim, podemos criar três tipos de popup boxes. São eles: Alert box, Confirm box e Prompt box.

Vamos conhecer cada um deles.

2.1 Alert box

Sintaxe

```
alert("Olá Mundo!");
```

Esse tipo de popup box tem o objetivo de alertar o usuário sobre algo que ocorreu durante alguma ação por ele executada.

Um botão de OK é exibido junto a ela. Para prosseguir com a navegação é necessário que o usuário clique nesse botão.

2.2 Confirm box

Sintaxe

```
confirm("Você deseja continuar?");
```

Com o Confirm Box, podemos ter a certeza se o usuário deseja ou não executar determinada ação, uma vez que esse box exibe dois botões: **OK** e **Cancel**.

Ao clicar em OK o box retorna true e ao clicar em Cancel o box retorna false.

Exemplo:

```
Ex.
if (confirm ("Algo está errado...devo continuar??"))
{  alert("Continuando") }
else
     { alert("Parando") }
```

2.3 Prompt Box

Sintaxe

```
Receptor = prompt ("Minha mensagem", "Meu texto")
```

Onde:

- Receptor é o campo que vai receber a informação digitada pelo usuário.
- Minha mensagem é a mensagem que vai aparecer como Label da caixa de input
- Meu texto é um texto, opcional, que aparecerá na linha de digitação do usuário

Exemplo.

```
var entrada = prompt("Informe uma expressão matemática", "")
var resultado = eval(entrada)
document.write("O resultado é = " + resultado)
```

Vejamos um exemplo integrando variável, condição, operadores, confirm box, alert box e prompt box:

```
var querMsg = confirm("Gostaria de receber uma mensagem de boas vindas?");
if(querMsg) {
       var nome = prompt("Por favor, qual o seu nome?","");
       if(nome != null && nome != "") {
             alert("Olá " + nome + ", seja bem vindo.");
```

```
            }
}
```

3. Loops

As declarações de looping em JavaScript, assim como em outras linguagens, tem por objetivo a execução de um mesmo bloco de código por uma determinada quantidade de vezes ou enquanto uma condição for verdadeira.

3.1 Declaração for

Looping que percorre determinado bloco de código por uma quantidade especificada de vezes.

Sintaxe

```
for (var = valorInicial; var <= valorFinal; var = var + incremento) {
        //código a ser executado.
}
```

Exemplo:

```
var numero = 1;
for (numero = 1; numero <= 5; numero++) {
        alert("O número atual é: " + numero);
}
```

3.2 Declaração while

Looping que percorre determinado bloco de código se e enquanto a condição for verdadeira.

Sintaxe

```
while (var <= valorFinal) {
        //código a ser executado.
}
```

Exemplo

```
var numero = 1;
while (numero <= 5) {
        alert("O número atual é: " + numero);
        numero = numero + 1;
}
```

3.3 Declaração do...while

Esse loop é uma variável do loop while. Ele sempre executará um bloco de código uma vez e o repetirá enquanto a condição for verdadeira. Mesmo que uma condição seja falsa, esse loop ocorrerá uma vez pois a condição é verificada após a execução.

Sintaxe

```
do {
       //código              a              ser              executado.
} while (var <= valorFinal);
```

Exemplo

```
var numero = 1;
do {
       alert("O número atual é: " + numero);
       numero = numero + 1;
} while (numero < 1).
```

3.4 Declaração break e continue

São declarações especiais para se trabalhar com looping. O break interrompe o looping e prossegue com o restante do código (se houver). O continue pára o loop atual e continua com o próximo valor.

Exemplo - break

```
var numero = 95;
for (numero = 95; numero < 150; numero++) {
       if(numero == 100) {
                break;
        }
        alert("Número: " + numero);
}
```

Neste exemplo, o loop *for* deveria ocorrer até o número 149 (*numero* < 150). Porém, utilizamos à condição *if* para detectar quando a variável *numero* estiver no número 100.

Quando isto ocorrer, utilizamos o comando *break* para interromper o loop, assim, chegaremos somente até o número 100.

Exemplo - continue

```
var idade = 20;
for (idade = 20; idade <= 25; idade++) {
      if(idade == 24) {
               continue;
       }
       alert("Você tem " + idade + " anos ?");
}
```

Conforme dito anteriormente, o comando *continue* interrompe o loop em um determinado momento e continua com o mesmo.

No exemplo acima, o loop *for* deveria rodar do número 20 até o 25, porém o comando *continue* interrompe a exibição do número 24.

4. Funções

As funções são blocos de códigos reutilizáveis. Elas são executadas através de algum evento ou quando são invocadas.

Você pode chamar uma função de qualquer lugar da página ou até mesmo de outras páginas contanto que o script esteja em um arquivo externo (.js).

As funções podem conter parâmetros ou não e, quando eles existem, são informados à função quando a chamamos.

Sintaxe - sem parâmetros

```
function nomeDaFuncao() {
        //códigos referente à função.
}
```

Sintaxe - com parâmetros

```
function nomeDaFuncao(varivel 1, variavel 2, variavel 3, ...) {
       //códigos referente à função.
}
```

Exemplo - sem parâmetros

```
function boasVindas() {
        alert("Olá, seja bem vindo.\nEspero que goste do nosso site.");
 }

 // Um exemplo para chamá-la.
 <a href="" onclick="javascript:boasVindas();">Chamar a Função</a>
```

Exemplo - com parâmetros

```
function boasVindas(nome) {
        alert("Olá " + nome + ", seja bem vindo.\nEspero que goste do nosso site.");
       }

 // Um exemplo para utilizá-la.
 <input type="button" value="Mensagem de boas vindas" onclick="boasVindas(document.nomeForm.nomeCampo.value);" />
```

Interpretamos essa declaração da seguinte forma: *document* é igual ao documento, ou seja, a página em si; *nomeForm* é igual ao nome do *form*, concedido a ele através do atributo *name* da tag *form*; *nomeCampo* é o nome do campo no qual o usuário informou o seu nome; já o *nomeCampo.value* é o valor contido no campo, no caso, o nome informado.

4.1 A declaração return

Quando queremos que uma função retorne algum valor, utilizamos a declaração *return*, a qual tem o papel de especificar o valor retornado pela função.

Exemplo

```
function multiplica(valor1,valor2) {
        total = valor1 * valor2;
        return total;
 }

 // Chamando a função através do alert().
 alert(multiplica(58,10));
```

Algumas importantes considerações sobre as funções:

- A palavra *function* sempre deve ser escrita com letras minúsculas;
- O nome da função pode conter letras minúsculas e/ou maiúsculas;
- As chaves { } indicam o início e o término da função;
- Mesmo que uma função não contenha parâmetros, os parentêses ( ) devem ser incluídos após o nome da função.

5. Eventos

Podemos definir um evento como uma ocorrência ou algum acontecimento realizado por uma página, uma ação do mouse, teclado, entre outros. Todos esses eventos podem ser detectados pelo JavaScript e conseqüentemente realizar alguma ação mediante tal acontecimento.

Em muitos dos casos, os eventos são utilizados em combinação com as funções. Isso significa dizer que a função não será executada até que o evento ocorra, por exemplo: a abertura de uma janela popup quando um link for clicado (onclick).

Vamos conhecer os eventos mais importantes e mais utilizados em nosso dia-a-dia.

5.1 onclick

O evento **onclick** ocorre quando o usuário clica sobre algum elemento da página, por exemplo: um link, uma imagem, um botão, dentre outros.

onclick na realidade quer dizer: pressionar o mouse (mousedown) e liberá-lo (mouseup) sobre um mesmo elemento.

onclick - Ocorre quando o objeto recebe um Click do Mouse. Válido para os objetos Buton, Checkbox, Radio, Link, Reset e Submit.

Exemplo:

- Função JavaScript:

```
function funcaoParaClick() {
 alert("Alerta exibido através do evento: onclick.");
}
```

- Chamada no HTML:

```
<a href="" onclick="javascript: funcaoParaClick();"> Clique aqui </a>
```

5.2 onload e onunload

O evento **onload** é executado quando uma página HTML é carregada por completa, incluindo as imagens. Já o evento **onunload** ocorre quando o usuário sai de uma página.

Eles são opostos entre si, enquanto um é utilizado na entrada o outro trabalha na saída.

onload - Ocorre na carga do documento. Ou seja, só ocorre no BODY do documento.

onunload - Ocorre na descarga (saída) do documento. Também só ocorre no BODY.

Exemplo:

- Função JavaScript:

```
function funcaoParaLoad() {
 alert("A página foi carregada por completo. Em consequência disso, este alerta foi exibido.\nO evento onload funciona dessa maneira.");
}

function funcaoParaUnLoad() {
 alert("Este alerta foi exibido através do evento: onunload. Que é executado, quando sairmos de uma página.");
}
```

- Chamada no HTML:

```
<body onload="funcaoParaLoad();" onunload="funcaoParaUnLoad();">
```

5.3 onmouseover e onmouseout

Esses eventos trabalham de forma oposta, porém são utilizados em conjunto na maioria das vezes.

Enquanto o evento **onmouseover** é acionado quando o mouse se localiza na área de um elemento, o **onmouseout** é executado quando ele (mouse) sai dessa área.

onmouseover - Ocorre quando o ponteiro do mouse passa por sobre o objeto. válido apenas para Link.

Exemplo:

- Chamada no HTML:

```
<a href="" onmouseover="funcaoParaMouseOver();" >MouseOver</a>
<a href="" onmouseout="funcaoParaMouseOut();" >MouseOut</a>
```

- Função JavaScript:

```
function funcaoParaMouseOver() {
```

```
      alert("Você PASSOU o mouse sobre a área do link. Então, exibimos este
alerta, através do evento: onmouseover.");
     }


     function funcaoParaMouseOut() {
      alert("Você RETIOU o mouse sobre a área do link. Então, exibimos este
alerta, através do evento: onmouseoout.");
     }
```

5.4 onfocus, onblur e onchange

Esses três eventos são utilizados na maioria das vezes em associação com algum elemento de formulário, sendo o **onfocus** e **onblur**antagônicos.

O evento **onfocus** ocorre quando o usuário clica em um link ou em um campo de texto e o foco é mantido até que outro elemento o ganhe. Já o **onblur** é executado quando o elemento perde o foco.

O exemplo mais clássico para o evento **onchange** (ao trocar/mudar) é quando alteramos uma opção na lista de um combobox. Nessa ação, ocorre o **onchange**.

onchange - Ocorre quando o objeto perde o focus e houve mudança de conteúdo. Válido para os objetos Text, Select e Textarea.

onblur - Ocorre quando o objeto perde o focus, independente de ter havido mudança. válido para os objetos Text, Select e Textarea.

onfocus - Ocorre quando o objeto recebe o focus. Válido para os objetos Text, Select e Textarea.

Exemplos

- Chamada no HTML:

```
<input type="text" onfocus="funcaoParaFocus();" />
<input type="text" onblur=""="funcaoParaBlur();" />
```

- Função JavaScript:

```
function funcaoParaFocus() {
 alert("Campo texto recebeu o foco");
}


function funcaoParaBlur() {
 alert("Campo texto perdeu o foco");
}
```

5.5 onsubmit

Para realizarmos a validação de um formulário utilizamos o evento **onsubmit**, o que significa dizer: ao enviar os dados do formulário.

Esse evento trabalha em conjunto com uma função da seguinte forma: associamos ao formulário a chamada de uma função e esta tem por objetivo validar os dados submetidos e retornar um valor verdadeiro (true) ou falso (false).

Se o valor retornado for verdadeiro, o formulário é enviado, caso contrário, o envio é bloqueado até que os dados sejam preenchidos como desejamos.

onsubmit - Ocorre quando um botão tipo Submit recebe um click do mouse. Válido apenas para o Form.

Exemplo

- Chamada no HTML:

```
... onsubmit="return confereFormulario();" ...
```

O evento onclick também pode chamar uma função para validar um formulário, assim como outros eventos, o onsubmit por ser o mais utilizado.

5.6 onkeydown, onkeypress e onkeyup

Esses três eventos são utilizados em associação com o teclado. Com eles podemos por exemplo: contar os caracteres de uma textarea, realizar o preview de algum texto, efetuar uma busca instantânea, dentre outras opções.

Os eventos **onkeydown** e **onkeypress** são semelhantes e ocorrem quando uma tecla é pressionada pelo usuário em seu teclado.

Já o **onkeyup** é executado quando a tecla é liberada, ou seja, ela foi pressionada e em seguida liberada.

Exemplos

```
... onkeydown="funcaoParaKeyDown();" ...
... onkeypress="funcaoParaKeyPress();" ...
... onkeyup="funcaoParaKeyUp();" ...
```

Exemplo onload:

- JavaScript:

```
function Cores(){
      horario=new Date()
      segundo=horario.getSeconds()
      if(segundo>=0 && segundo<10){document.bgColor="yellow"}
      if(segundo>=11 && segundo<20){document.bgColor="red"}
      if(segundo>=21 && segundo<30){document.bgColor="blue"}
      if(segundo>=31 && segundo<40){document.bgColor="orange"}
      if(segundo>=41 && segundo<50){document.bgColor="green"}
      if(segundo>=51 && segundo<60){document.bgColor="gray"}
}
window.setInterval("Cores()",1000)
```

- HTML

```
<body onLoad="Cores();">
```

Exemplo onmouseover:

- JavaScript:

```
function Trocafoto2(){
      document.images["foto"].src="flores2.jpg"
}
function Trocafoto3(){
      document.images["foto"].src="flores3.jpg"
}
function Trocafoto4(){
      document.images["foto"].src="flores4.jpg"
}
//essa função retorna a imagem inicial
function Voltafoto1(){
      document.images["foto"].src="flores1.jpg"
}
```

- HTML

```
<div><img src=flores1.jpg name=foto width=300 height=250></div>
<div><img  src=flores2.jpg  width=100  height=100  onMouseOver="Trocafoto2()"
onMouseOut="Voltafoto1()"></div>
<div><img  src=flores3.jpg  width=100  height=100  onMouseOver="Trocafoto3()"
onMouseOut="Voltafoto1()"> </div>
<div><img  src=flores4.jpg  width=100  height=100  onMouseOver="Trocafoto4()"
onMouseOut="Voltafoto1()"></div>
```

6. Array

O objeto Array é definido através da palavra-chave *new*.

Vejamos um exemplo:

```
var diasDaSemana = new Array();
var diasDaSemana = new Array(7);
```

Ambos os exemplos acima estão corretos. A diferença entre eles é que o segundo informa o tamanho (valor total) que o Array terá através do número inteiro (integer) 7 (sete).

Agora, vamos conhecer duas formas para criarmos um Array contendo os dias da semana.

```
var diasDaSemana = new Array();
diasDaSemana[0] = "Domingo";
diasDaSemana[1] = "Segunda";
diasDaSemana[2] = "Terça";
diasDaSemana[3] = "Quarta";
diasDaSemana[4] = "Quinta";
diasDaSemana[5] = "Sexta";
diasDaSemana[6] = "Sábado";
```

```
var diasDaSemana = new Array("Domingo", "Segunda", "Terça", "Quarta",
"Quinta", "Sexta", "Sábado");
```

Tanto os valores contidos na primeira forma quanto na segunda, são acessados assim:

```
alert(diasDaSemana[6]); // Exibirá: Sábado.
```

Portanto, para referenciarmos a um valor dentro de um Array, basta indicarmos o nome do Array e o índice númerico do valor. É importante lembrar que o índice númerico se inicia com 0 (zero).

Se você especificar algum número ou os valores true/false (verdadeiro/falso) como o valor de um Array, o tipo da variável será númerica e Booleana respectivamente, ao invés de string.

Os métodos do objeto Array()

Relação dos métodos mais utilizados.

Método	Descrição
join()	Coloca todos os elementos de um Array em uma string e os separam por um delimitador especificado.
shift()	Retorna o primeiro valor de um Array e remove os demais.
pop()	Retorna o último elemento de um Array e remove os demais.
reverse()	Inverte a ordem dos elementos de um Array.
slice()	Retorna os elementos selecionados de um Array.
sort()	Ordena os elementos de um Array.

7. Data

Para trabalharmos com datas e horários em JavaScript fazemos uso do objeto **Date()** e, para o instanciarmos (criarmos objetos), utilizamos a palavra-chave *new*.

Vejamos um exemplo:

```
var dataAtual = new Date();
alert(dataAtual);
```

No exemplo acima teremos a data e o horário atual, ou seja, o exato momento em que o script foi executado.

Podemos também manipular esse formato para podermos exibí-lo como desejarmos.

Veja um novo exemplo adaptado ao nosso formato *dd/mm/aaaa*.

Exemplo

```
var data = new Date();
var mes = new Array(12);
mes[0] = "Janeiro";
mes[1] = "Fevereiro";
mes[2] = "Março";
```

```
mes[3] = "Abril";
mes[4] = "...";
alert(data.getDate() + "/" + mes[data.getMonth()] + "/" + data.getFullYear());
```

No exemplo acima utilizamos o objeto **Date** e três de seus métodos: **getDate()**, **getMonth()**, **getFullYear()**.

O primeiro retorna o dia do mês (de 1 a 31), o segundo o mês (de 0 a 11) e o último o ano com quatro digítos.

O diferencial acima está na utilização do Array para o mês, uma vez que o método **getMonth()** retorna números de 0 (zero) a 11 (onze) e, assim, o número 4 (quatro) representaria o mês de maio. Diferentemente de nossa representação usual, onde esse mês é representado pelo algarismo 5 (cinco).

Assim, utilizamos um Array para setarmos o nome do mês (pode ser algarismos também) de acordo com o número retornado pelo método.

Os métodos do objeto Date()

Relação dos métodos mais utilizados.

Método	**Descrição**
Date()	Retorna a data e o horário atual.
getDate()	Retorna o dia do mês (1-31).
getDay()	Retorna o dia da semana (0-6).
getMonth()	Retorna o mês (0-11)
getFullYear()	Retorna o ano com quatro dígitos.
getHours()	Retorna a hora (0-23).
getMinutes()	Retorna os minutos (0-59).
getSeconds()	Retorna os segundos (0-59).
getMilliseconds()	Retorna os milisegundos (0-999).
getTimezoneOffset()	Retorna a diferença em minutos entre o tempo local e o do Meridiano de Greenwich (GMT)

Os métodos acima precisam atuar em conjunto com o objeto **Date()** porque que eles são métodos desse objeto.

Sua utilização é da seguinte forma:

```
// Uma forma
new Date().nomeDoMetodo();
// Outra forma
var data = new Date();
data.nomeDoMetodo();
```

8. Trabalhando com Janelas

Podemos abrir novas janelas sobre uma janela contendo o documento principal. A sintaxe geral deste método é a seguinte:

```
Variavel = window.open ("Url", "Nome da janela", "Opções")
```

Onde:

Variavel - Nome que será atribuído como propriedade da janela.

Url - Endereço Internet para ser aberto na

Nome da Janela - É o nome que aparecerá no top da janela (Título)

Opções - São as opções que definem as características da janela, quais sejam:

toolbar - Cria uma barra de ferramentas tipo "Back", "Forward", etc.

location - Abre a barra de location do browse

directories - Abre a barra de ferramentas tipo "What's New", "Handbook", etc.

status - Abre uma barra de status no rodapé da janela

scrollbars - Abre barras de rolamento vertical e horizontal

menubar - Cria uma barra de menu tipo "File", "Edit", etc.

resizable - Permite ao usuário redimencionar a janela

width - Especifica a largura da janela, em pixels

height - Especifica a altura da janela, em pixels

Todas as opções (exceto width e height) são boleanas e podem ser setadas de duas formas.

Exemplo: "toolbar" ou "toolbar=1" são a mesma coisa.

Se nada for especificado, entende-se que todas as opções estão ligadas; Caso seja especificada qualquer opção, será entendido que estão ligadas apenas as opções informadas. As opções devem ser informadas separadas por vírgula, sem espaço entre elas.

Exemplo:

- Função JavaScript:

```
function AbreJanela(endereco)
{
popup = window.open(endereco,"Detalhes","toolbar=0,scrollbars=1,location=0,directories=0,copyhistory=0,status=0,menubar=0,resizable=1,width=550,height=450,z-lock,screenX=90,screenY=100, Left=120, Top=50");
}
```

- Chamada no HTML:

```
<a href=# onclick="AbreJanela('data.html');"> Abrir janela </a>
```

Para fechar a janela, utilize o seguinte método:

```
Variavel.document.write ("window.close()")
```

- Função JavaScript:

```
popup.document.write ("window.close()")
```

9. String

O objeto string é utilizado para manipular um texto armazenado em uma variável e, para essa manipulação, ele nos disponibiliza diversos métodos.

É importante ressaltar que, além do objeto string, há a string literal. Quando utilizamos alguns dos métodos do objeto string na string literal, essa última é convertida automatica e temporariamente em um objeto string. Por exemplo, o método *length* retorna a quantidade de caracteres em uma string.

Veja a utilização desse método abaixo em ambos os casos:

```
var variavel1 = "Teste";

var variavel2 = new String("Teste");

alert("A variável variavel1 contém: " + variavel1.length + " caracteres.");

alert("A variável variavel2 contém: " + variavel2.length + " caracteres.");
```

No exemplo acima, teremos o valor 5 (cinco) nos dois casos (string literal - variavel1; objeto string - variavel2) porque 5 é o total de caracteres contido no nome Teste.

Os métodos do objeto String()

Relação dos métodos mais utilizados.

Método	Descrição
indexOf()	Retorna a posição da primeira ocorrência de um valor especificado em uma string.
lastIndexOf()	Retorna a posição da última ocorrência de um valor especificado em uma string.
match()	Procura por um valor específico em uma string. Se encontrado, ele (valor) é retornado, caso contrário, retorna *null*.
replace()	Substitue alguns caracteres por outros caracteres em uma string.
toLowerCase()	Exibe os caracteres da string em minúsculos.
toUpperCase()	Exibe os caracteres da string em maiúsculos.
substr()	Extrai uma quantidade específica de caracteres de uma string a partir de um índice inicial.
substring()	Extrai os caracteres de uma string entre dois índices.

10. Math

O objeto Math é utilizado para realizar tarefas matemáticas fornecendo-nos diversos valores e funções.

Para se trabalhar com o objeto Math não é necessário defini-lo, ao contrário dos outros objetos que conhecemos.

Valores matemáticos

Relação dos valores matemáticos que podem ser acessados pelo objeto Math.

Math	Nome	Valor
Math.E	E	2.718281828459045
Math.PI	PI	3.141592653589793
Math.SQRT2	square root of 2	1.4142135623730951
Math.SQRT1_2	square root of 1/2	0.7071067811865476
Math.LN2	natural log of 2	0.6931471805599453
Math.LN10	natural log of 10	2.302585092994046
Math.LOG2E	base-2 log of E	1.4426950408889634
Math.LOG10E	base-10 log of E	0.4342944819032518

10.1. Métodos (funções) matemáticos

ceil()

Arredonda o valor informado para cima até o número inteiro mais próximo. Veja o exemplo:

```
var numero = 6.1;
alert(Math.ceil(numero));
// Exibirá o valor 7.
```

floor()

Arredonda o valor informado para baixo até número inteiro mais próximo. Veja o exemplo:

```
var numero = 6.1;
alert(Math.floor(numero));
// Exibirá o valor 6.
```

max()

Retorna o número de maior valor entre dois números especificados.

```
var numero1 = 6.7;
var numero2 = 3.8;
alert(Math.max(numero1,numero2));
```

```
// Exibirá o valor 6.7
```

min()

Retorna o número de menor valor entre dois números especificados.

```
var numero1 = 6.7;
var numero2 = 3.8;
alert(Math.min(numero1,numero2));
// Exibirá o valor 3.8
```

random()

Retorna um número entre 0 e 1.

```
alert(Math.random());
// Exibirá números aleatórios entre 0 e 1 - Ex.: 0.8335737463859384
alert(Math.random()*59);
// Exibirá números aleatórios entre 0 e 59 - Ex.: 58.92792655560298
```

round()

Arredonda o valor informado para o número inteiro mais próximo, seja positivo ou negativo.

```
var numero1 = 6.7;
alert(Math.round(numero1));
// Exibirá o valor 7
var numero2 = -3.8;
alert(Math.round(numero2));
// Exibirá o valor -4
```