1.0 Introdução à JavaScript

JavaScript é uma linguagem para auxílio na criação de Home-Pages, as funções escritas em JavaScript podem ser embutidas dentro de seu documento HTML, possibilitando o incremento das funcionalidades do seu documento HTML com elementos interessantes. Sendo possível: responder facilmente a eventos iniciados pelo usuário, incluir efeitos que tornem sua página dinâmica. Logo, podemos criar sofisticadas páginas com a ajuda desta linguagem.

1.1 Introdução ao JavaScript

JavaScript é uma linguagem de programação voltada para ambiente web que tem como objetivo "interagir" com a página HTML.

As páginas HTML foram construídas após a criação da internet com o objetivo de divulgar notas científicas e artigos técnicos entre as universidades norte-americanas.

A metodologia aplicada no desenvolvimento e criação da linguagem HTML informava que o conteúdo das páginas HTML seria estático. Eles não tinham a visão que a internet teria um papel tão grande no amadurecimento da sociedade.

Inicialmente JavaScript não era JavaScript, mas sim LiveScript, uma tecnologia criada pela Netscape, quando ela estava desenvolvendo a versão

2.0 de seu navegador. Ela se destacou principalmente pelo fato de não precisar ser compilada (interpretada), sendo adicionada no documento HTML e de seus scripts não serem lentos, uma vez que eles não precisavam ser executados no servidor, mas sim no PC do usuário que estava acessando a página. Na época, não houve muito interesse pelo LiveScript, pois havia um grande marketing em relação a linguagem Java, criada pela Sun Microsystems há 3 anos e que oferecia um potencial muito maior que o LiveScript.

Para aproveitar deste marketing, a Netscape tornou o Netscape 2.0 compatível com a linguagem Java e deu apoio a Sun Microsystems para reestruturar o LiveScript de acordo com a linguagem Java. Naquele momento, o LiveScript se tornou JavaScript.

A consequência desta mudança foi em criar uma linguagem de script que conseguiu unir os dois

universos: um simples como LiveScript e outro robusto, poderoso e "conhecido" como Java.

A partir da versão 2.0 do Netscape e da versão 3.0 do Internet Explorer, todo navegador web suporta JavaScript e desde então o JavaScript tem ganhado grandes evoluções. Hoje o JavaScript é um padrão aberto e empresas de todo o mundo o suportam.

1.2 Java é diferente de JavaScript

Apesar dos nomes bem parecidos, Java não é o mesmo que JavaScript. Estas são duas técnicas diferentes de programação na Internet. Java é uma linguagem de programação. JavaScript é uma linguagem de hipertexto.

A linguagem de programação JavaScript, desenvolvida pela Netscape, Inc., não faz parte da plataforma Java.

JavaScript não cria applets ou aplicativos independentes. Em sua forma mais comum hoje, JavaScript reside dentro de documentos HTML e pode fornecer níveis de interatividade com páginas da Web que não podem ser conseguidos com HTML simples.

As diferenças principais entre Java e JavaScript estão listadas.

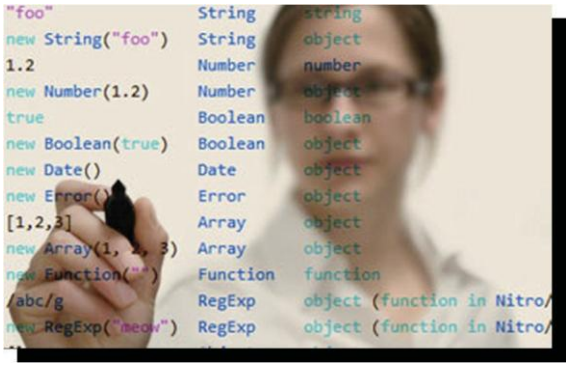
Java é uma linguagem de programação OOP, ao passo que Java Script é uma linguagem de scripts OOP.

Java cria aplicativos executados em uma máquina virtual ou navegador, ao passo que o código JavaScript é executado apenas em um navegador.

O código Java precisa ser compilado, ao passo que os códigos JavaScript estão totalmente em texto, apenas interpretado.

Eles requerem plug-ins diferentes.

1.3 Entendendo o JavaScript

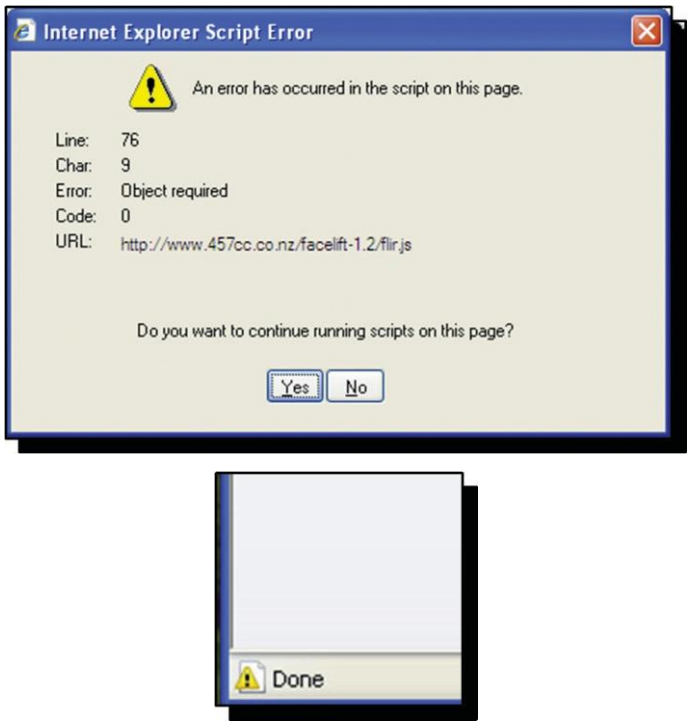


Por que o JavaScript é tão rápido? Acredito que esta seja uma das perguntas que a maioria dos desenvolvedores fazem. Para toda uma pergunta simples há uma resposta simples: Ela não é processada no servidor!

JavaScript não é processada no servidor por que ela não é uma linguagem compilada, mas sim interpretada. Quem a interpreta não é o servidor, mas sim o navegador do usuário que acessa a página

HTML. Então, todo o processamento dos scripts em JavaScript fica por conta do PC do usuário. Quanto mais rápido ele for, mais rápido será o processamento.

Para ter certeza disso, execute uma página HTML (que execute uma função em JavaScript) que está no seu próprio PC e acesse esta mesma página em um servidor web. Você irá perceber que a diferença de performance é muito pequena, pois o tempo que o navegador irá perder será o de baixar a página HTML, juntamente com suas dependências (em JavaScript ou não) para depois executá-la.



Já são embutidas no navegador do PC do usuário as bibliotecas de run-time para que ele entenda os scripts em JavaScript. Caso você use um navegador anterior ao Netscape 2.0 e ao Internet Explorer 3.0, provavelmente você não irá conseguir executar os scripts escritos em JavaScript, tampouco em qualquer outra linguagem. Erros no JavaScript

Como o JavaScript é uma linguagem interpretada, todo e qualquer erro será detectado na hora em que o navegador estiver lendo e/ou executando os scripts em JavaScript. Para verificar se seu script está errado ou houve algum erro de execução, basta verificar a barra de status do seu navegador. Caso ela tenha um ícone de aviso, informando um texto parecido ou semelhante com "Erro na consulta", clique duas vezes no ícone de aviso para visualizar os erros.

1.4 Usando JavaScript

Um código JavaScript pode ser inserido em um documento HTML de duas formas:

Colocando o código JavaScript como filho de um elemento com a tag script;

```
<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="utf-8">
    <title>
        Inserindo código JS em um documento HTML
    </title>
    <script type="text/javascript" src="codigo.js">
    </script>
</head>

<body>

</body>
</html>
```

Utilizando o atributo src de um elemento com a tag script no qual devemos passar o caminho relativo ou absoluto para um arquivo que contenha o código JavaScript.

```
<!DOCTYPE html>
<html lang="pt">
    <head>
        <meta charset="utf-8">
        <title>
            Inserindo código JS em um documento HTML
        </title>
        <script type="text/javascript">
            window.onload=function() {
                document.getElementById("ola-mundo").
                    innerHTML='Olá Mundo!';
            }
        </script>
    </head>

    <body>
        <p id="ola-mundo"> </p>
    </body>
</html>
```

Exercícios Propostos

Como aconteceu a evolução da tecnologia JavaScript no decorrer dos anos?

Qual a diferença entre Java e JavaScript?

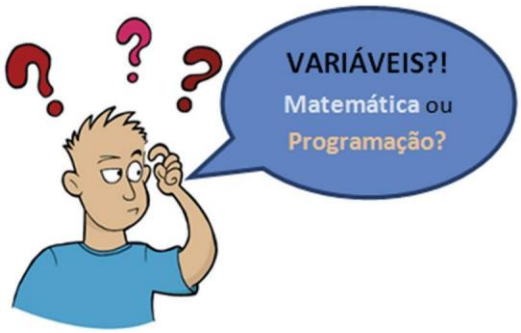
Quais as formas de se trabalhar com JavaScript junto com html. Dê exemplos.

Como funciona o processamento do JavaScript em uma página web?

Qual a importância do JavaScript em um projeto Web?

Pesquise e cite quais frameworks para JavaScript existem atualmente no mercado.

2.0 Variáveis



Sim, variáveis.

Na programação, a utilização da matemática é constante e é muito importante que você tenha um bom raciocínio lógico. Lembra daquela matéria de lógica matemática que você estudou no ensino fundamental e médio, achando que nunca iria utilizar o que estava aprendendo? Adivinha, vamos precisar daqueles conhecimentos agora!

2.1 Mas o que são variáveis em programação?

Vamos entender variável, como uma caixa, na qual você pode dar o nome que lhe achar conveniente, e guardar o conteúdo que desejar.

Ou seja, toda variável tem um nome, valor e tipo.

2.2 Mas o que seria o tipo de uma variável?

O tipo é uma classificação que damos a variável, essa classificação informa qual forma de dado se encontra ali armazenado.

As variáveis, podem ser classificadas com os tipos:

- Inteiro: Todo e qualquer dado numérico que pertença ao conjunto de números inteiros relativos (negativo, nulo ou positivo). Exemplos: {...-4,-3,-2,-1,0,1,2,3, 4...}.
- Real: Todo e qualquer dado numérico que pertença ao conjunto de números reais (negativo,

nulo ou positivo). Exemplos: {15.34; 123.08; 0.005 -12.0; 510.20}.

- Numérico: Trata-se de todo e qualquer número que pertença ao conjunto dos inteiros ou reais, também abrange números binários, octal e hexadecimal.
- Caracteres: são letras isoladas. Exemplo: {'a', 'd', 'A', 'h'}.
- Dados literais: Conhecido também como Conjunto de caracteres ou String, é todo e qualquer dado composto por um conjunto de caracteres alfanuméricos (números, letras e caracteres especiais). Exemplos: {“Aluno Aprovado”, “10% de multa”, “Confirma a exclusão ??”, “S”, “99-3000-2”, “email”,”123nm”,”fd54fd”}.
- Lógico: A existência deste tipo de dado é, de certo modo, um reflexo da maneira como os computadores funcionam. Muitas vezes, estes tipos de dados são chamados de booleanos, devido à significativa contribuição de Boole a área da lógica matemática. A todo e qualquer dado que só pode assumir duas situações dados biestáveis, algo como por exemplo {0/ 1, verdadeiro/falso, sim/não, true/false}.

Então, o tipo de uma variável é determinado pelo conteúdo que ela guarda. Se uma variável armazena um número inteiro ela é do tipo inteiro, se é um conjunto de caracteres ela é do tipo String.
Dependendo da linguagem de programação que você estiver programando, os nomes dos tipos listados acima podem mudar e novos tipos também poderão surgir.

2.3 Nomeando variáveis

No tópico anterior vimos que as varáveis precisam de nome, tipo e valor. Já estudamos um pouco sobre tipos de variáveis, agora iremos discutir sobre como criar um nome ou **identificador** válido para uma variável.
As regras para se criar um identificador de variável podem sofrer algumas alterações dependendo da linguagem de programação adotada vamos listar as regras que geralmente são adotadas pela maioria:

Os identificadores são **case sensitive**, ou seja, faz distinção entre maiúsculas e minúsculas. Isso significa que um nome de variável, como **meuContador**, é diferente do nome de variável **MEUCountador**;
Nomes de variáveis podem ser de qualquer comprimento, em algumas linguagens existe limite no número de caracteres que compõe o identificador, por essa e pela questão de legibilidade tente não ultrapassar 15 caracteres.
Crie identificadores que informe explicitamente para que aquela variável irá servir na composição do seu programa, ou seja se uma variável vai armazenar um valor inteiro que irá aumentar de um em um até chegar 100, coloque o nome dessa variável de **contador**
O primeiro caractere deve ser uma letra ASCII (em maiúscula ou minúscula) ou um caractere de sublinhado (_). Observe que um número não pode ser usado como o primeiro caractere.

Os caracteres subsequentes devem ser letras, números ou sublinhados (_).

O nome da variável não deve ser uma **palavra reservada**. Existem palavras que são utilizadas pela linguagem de programação, que você está utilizando para implementar seus programas, para controlar e estruturar seu algoritmo. Essas palavras classificadas com o nome de **palavras reservadas**.

Questão: Quais dos identificadores de variáveis abaixo são válidos?

_pagecount

99Balloons

Number_Items

Alpha&Beta

Part9

2.4 Declarando ou criando uma variável

Declarar uma variável tecnicamente é pedir ao sistema operacional um espaço exclusivo na memória RAM do seu computador para armazenar uma informação.

Abaixo temos exemplos de declaração e inicialização de variáveis em linguagens de programação distintas:

String nome = new String(“Jonas”);

int idade = 25;

char opcao = ‘S’;

Inteiro quantidade = 100;

contador = 1;

modeloCarro = “Celta”;

Perceba que dependendo da linguagem de programação processo de declaração é diferente.

2.5 Linguagens de Programação Fracamente e Fortemente Tipadas

Linguagem fortemente tipadas(LST), são linguagens de programação que cada variável do programa, representa um tipo bem definido, ou seja explicitamente você será obrigado a declarar o tipo da variável ao qual estará declarando e caso tente fazer operações com variáveis de tipos diferentes precisará fazer conversões, caso contrário irá acontecer erros em seu código.

Exemplo de declaração de variáveis fortemente tipadas:

Inteiro contador = 1;

Caracter sexo = ‘F’;

String nome = “Jonas”;

Já nas linguagens de programação que são classificadas como Linguagem de Programação Fracamente

Tipada(LWT) ou de tipagem dinâmica, não precisamos colocar o tipo da variável antes de seu identificador, além do tipo da variável ser modificado em tempo de execução sem precisar fazer conversão, ou seja o tipo da linguagem é modificada automaticamente segundo o valor que você armazena na mesma.

idade = “18” - (tipo String);

idade = 18 - (agora a varável idade é do tipo inteiro);

2.6 Variáveis em JavaScript

Como a linguagem de programação escolhida para se aprender lógica de programação foi o JavaScript, vamos estudar nesse tópico como funciona a manipulação de variáveis nessa linguagem.

2.6.1. Declarando ou criando uma variável JavaScript

Diferente da maioria das linguagens o JavaScript define as variáveis dinamicamente, portanto ao atribuir uma variável ele escolhe o tipo conforme o valor passado para a variável, não sendo necessário especificar o mesmo.

Existem dois tipos de sintaxe de declaração de variável em JavaScript que são:

```
var nome-da-variável = valor-da-variável;   ou   nome-da-variável = valor-da-
```

Lembre-se que nome-da-variável precisa seguir todas as regras do tópico nomeando variáveis. Abaixo palavras reservadas da linguagem de programação JavaScript:

Palavras-chave em JavaScript				
break	case	catch	continue	default
delete	do	else	false	finally
for	function	if	in	instanceof
new	null	return	switch	this
throw	true	try	typeof	var
void	while	with		
Palavras-chave que são reservadas, mas não usadas pelo JavaScript				
abstract	boolean	byte	char	class
const	debugger	double	enum	export
extends	final	float	goto	implements
import	int	interface	long	native
package	private	protected	public	short
static	super	synchronized	throws	transient
volatile				

2.7 Tipos de variáveis JavaScript

Como já mencionado, no JavaScript não declaramos explicitamente o tipo de uma variável. Por exemplo, se a variável recebe um número inteiro declaramos, por exemplo, ***int x***, ou se ela recebe um número real declaramos ***float y***, mas no JavaScript isso não acontece. O tipo de uma variável é definido dinamicamente, não precisando o desenvolvedor especificar o seu tipo.

Veremos nos tópicos abaixo como trabalhar com os tipos de varáveis JavaScript.

2.7.1 Trabalhando com String e Números em JavaScript

Vamos ver como declaramos Número e String no JavaScript abaixo:

```
<script>
    var variavel = 10;           //Inteiro
    alert(variavel);

    var variavel2 = "João";      //String
    alert(variavel2);

    var variavel3 = 'Cristiane'; //String
    alert(variavel3);
</script>
```

Quando declaramos uma String utilizamos aspas duplas (") ou simples ('). Você pode declarar número utilizando aspas, mas não é obrigatório. Se utilizarmos no início de uma String a (“) temos que finalizarmos ela com a (”), isso vale para quando utilizamos (') .

2.7.2 Trabalhando com outros tipos de dados JavaScript

Agora, vamos descrever outros tipos de dados ou variáveis suportadas no JavaScript.

Booleanos: suportam dois valores true/false (verdadeiro ou falso). Exemplo:

```
<script>
    var variavel = false  //Boleano
    var variavel = true  //Boleano
</script>
```

Constante: Valor que não se altera no decorrer do código. Exemplo:

```
<script>
    const variavel = 7;
    alert(variavel);
</script>
```

Undefined: quando você tenta acessar um atributo de um objeto que não existe ou uma variável que ainda não foi inicializada o JavaScript define o valor dela como undefined.

Null: é um tipo especial de valor que indica para o navegador que a variável está vazia. O valor null que significa ausência de valor, vazio ou nada quando atribuído a uma variável está definindo-a como vazia. A diferença do null para undefined é que se uma variável tem o valor null ela está definida.

Array: o array é um tipo composto de dado. O array é uma estrutura de dados que possibilita guardar vários dados e indexa-los utilizando índices. Ou seja, uma variável que pode conter vários valores diferentes.

```
<script>
    var variosTipos = ["e-Jovem", 10, [1,2]];
</script>
```

Object: no JavaScript também podemos trabalhar com variáveis do tipo objeto, que são variáveis robustas onde podemos guarda desde um único valor até um programa completo. Iremos discutir sobre esse tipo em cursos posteriores. Veja abaixo um exemplo de criação de uma variável do tipo objeto no JavaScript.

```
<script>
    nome = new Array(3);
    nome[0]="André ";
    nome[1]="Thiago ";
    nome[2]="Nardy";
    document.write("Meu nome é " + nome[0] + nome[
        1] + nome[2])
</script>
```

Exemplo:

2.8 Variáveis Globais e Locais

As variáveis são classificadas em dois tipos em relação a sua área de atuação, que são as globais e as locais. A diferença entre elas é:

Variável Global: Criada ou declarada fora de uma função, portanto podem ser utilizadas a qualquer momento no seu script;

Variável Local: Criada ou declarada dentro de uma função, portanto só podem ser utilizadas dentro da função criada.

Funções são bloco de códigos que são executados todas as vezes que invocados por um identificador. Iremos discutir mais sobre funções em capítulos posteriores.

Exercícios Propostos:

Qual a principal finalidade de uma variável?

O que significa tipagem dinâmica.

Das variáveis abaixo, quais possuem nomenclaturas válidas na linguagem JavaScript.

a_____b;	a_1_;	_início;	@nome;	val_!;
nome;	a_|_;	#valor;	palavra;	tele#;
123;	_ = ;	VALOR_MAIOR;	_____;	all;

Crie dez variáveis atribuindo valores diversos, logo após use o comando **document.write()** pra imprimir na tela do browser, exemplo:

<script>

var nome = “Maria Cavalcante”; document.write(nome);

</script>

Quais os tipos de variáveis que podemos citar em JavaScript.

Como podemos distinguir um tipo de variável de outro, uma vez que a tipagem é feita de forma dinâmica no JavaScript.

Qual a principal finalidade de um constante e como elas são definidas em JavaScript.

Em que momentos precisamos converter uma variável de um tipo em outro. Crie uma constante e imprima com o comando document.write();

3.0 Operadores em JavaScript

Neste capítulo iremos estudar os tipos e quais os operadores; Falar do conceito de atribuição e concatenação de String; exemplificar os operadores, sua importância e funcionamento

Os operadores têm seu papel importante dentro de qualquer linguagem de programação. É através deles que podemos realizar diversos operações dentro de um programa, seja ela de atribuição, aritmética, relacional, lógico, dentre outros. Em JavaScript não é diferente, os operadores são utilizados constantemente, porém existem algumas regras que veremos mais adiante.

3.1 Operadores de strings

São operadores utilizados para unir o conteúdo de uma string a outra, com isso podemos dizer que há dois operadores de string. O primeiro é o operador de concatenação ('+') que já utilizamos em exemplos anteriores, ele retorna a concatenação dos seus argumentos direito e esquerdo. O segundo é o operador de atribuição de concatenação ('+='), que acrescenta o argumento do lado direito no argumento do lado esquerdo.

Observe o exemplo abaixo:

Nesse exemplo pode-se observar a declaração da variável **d**, logo após temos uma inicialização e atribuição de concatenação em uma mesma linha, isso é possível em JavaScript, deixando o código mais otimizado, porém menos legível.

3.2 Operadores Matemáticos

3.2.1 Aritméticos

Chamamos de **operadores aritméticos** o conjunto de símbolos que representa as operações básicas da matemática.

Operações	**Operadores**	**Exemplo**	**Resposta**
Adição	+	3 + 5	8
Subtração	-	20 – 5	15
Multiplicação	*	7 * 8	56
Divisão	/	15 / 3	5
Módulo (Resto da divisão)	%	10 % 3	1
Negação (Numero Posto)	-(valor)	Se for 15 a atribuição	-15

3.2.2 Atribuição

O operador básico de atribuição é "=" (igual). Com ele podemos atribuir valores as variáveis como foi visto em exemplos anteriores. Isto quer dizer que o operando da esquerda recebe o valor da expressão da

```
<script>
    a = (b=4) + 5;
    document.write("a = "+a+", b="+b);
</script>
```

direita (ou seja, "é configurado para"). Mas podemos usar algumas técnicas, observe o exemplo abaixo:

Resultado: a= 9, b=4

Além do operador básico de atribuição, há "operadores combinados" usados para array e string, eles permitem pegar um valor de uma expressão e então usar seu próprio valor para o resultado daquela expressão.

Por exemplo:

```
<script>
   //EXMPLO 01
    a  = 3;
    a *= 5; /* mesmo que: a = a*5, ou seja,
               a recebe seu valor anterior(3)
               e multiplica este valor por 5 */
   document.write("a = "+a);
   //EXMPLO 02
    b= "Bom ";
    b+="Dia!";
    document.write(" b = "+b);
</script>
```

Resultado: a = 15, b = Bom Dia!

Observe a expressão: a = 3 e logo após a*=5. Isto significa a mesma coisa de a = a * 5, ou, a = 3 *5. A ideia pode ser usada para string, como foi feito com a variável b, onde b = “Bom”, logo após usamos ponto(+) e igual(=) para concatenar os valores, ficando assim: b+=”Dia!”. Lembrando que isso significa a mesma coisa que b = b+“Dia”. Observe mais um exemplo:

```
<script>
    a  = "Dia ";
    b = "Bom ";
    b += a +="turma!"; //concatena 'a', depois 'b'
    document.write(" b = "+b);
</script>
```

Resultado: Bom Dia turma!

Podemos definir uma sequência com duas concatenações, onde a = “Dia”+“turma!” e logo após temos b = “Bom”+“Dia turma!”.

Os operadores de atribuição são usados para economizar linhas de código, deixando assim o código mais funcional e otimizado. A tabela abaixo mostra os principais operadores de atribuição:

Operadores	**Descrição**
=	*Atribuição simples.*
+=	*Soma, depois atribui. Quando usado com Strings concatena.*
-=	*Subtrai, depois atribui.*
*=	*Multiplica, depois atribui.*
/=	*Divide, depois atribui.*
%=	*Modulo(resto) da divisão, depois atribui.*

Observe um exemplo aplicando os operadores.

```
<script type="text/javascript">
    a = 8;
    // 'a' recebe seu valor anterior(8) e multiplica por 3;
    document.write( a*= 3);
    document.write( "<br/>");

    // 'a' recebe seu valor anterior(24) e divide por 2;
    document.write( a/= 2);
    document.write( "<br/>");

    // 'a' recebe o resto da divisão de 'a'(12) por 5;
    document.write( a%= 5);
    document.write( "<br/>");

    // 'a' recebe seu valor anterior(2) e soma com 3;
    document.write( a+= 3);
    document.write( "<br/>");

    // 'a' recebe seu valor anterior(5) e subtrai 1;
    document.write( a-= 1);
    document.write( "<br/>");
</script>
```

Resultado : 24 - 12 - 2 - 5 - 4

Vale ressaltar que a cada **document.write(),** o valor de **a** sofre modificações. Isso devido a atribuição feita após a operação. Usamos o operador ponto (+) para concatenar os valores obtidos com **
** código usado em HTML para quebra de linha.

3.3 Operadores de decremento e incremento

São operadores usados para atribuir em 1 ou -1 a variável, isso pode ser feito antes ou depois da execução de determinada variável. A tabela abaixo mostra tais operadores:

Operadores	Descrição
++a	*Pré-incremento. Incrementa a em um e, então, retorna a.*
a++	*Pós-incremento. Retorna a, então, incrementa a em um.*
- -a	*Pré-decremento. Decrementa a em um e, então, retorna a*
a- -	*Pós-decremento. Retorna a, então, decrementa a em um*

Exemplo:

```
<script type="text/javascript">
    a = 1;
    // 'a'(1) recebe o incremento de 1, e depois imprime seu valor(2);
    document.write( ++a);
    document.write( "<br/>");

    // 'a'(2) imprime primeiro seu valor(2), depois recebe o incremento de 1;
    document.write( a++);
    document.write( "<br/>");

    // 'a'(3) recebe o decremento de 1, e depois imprime seu valor(2);
    document.write( --a);
    document.write( "<br/>");

    // 'a'(2) imprime primeiro seu valor(2), depois recebe o decremento de 1;
    document.write( a--);
    document.write( "<br/>");

    // valor final de 'a'
    document.write(a);
</script>
```

Resultado : 2 2 2 2 1

Nesse exemplo temos uma forma aplicada do uso de decremento e incremento, lembrando que a variável a pode ter qualquer nome. Também podemos fazer um comparativo com o pré-incremento ou incremento-prefixado com operações que já conhecemos, observe:

Operador	Forma extensa.	Forma simplificada
++a	a = a + 1	a+=1
- -a	a = a + 1	a -=1

3.4 Operadores relacionais

Os operadores relacionais ou conhecidos também como operadores de comparação, são utilizados para

fazer determinadas comparações entre valores ou expressões, resultando sempre um valor booleano verdadeiro ou falso(TRUE ou FALSE). Para utilizarmos esses operadores usamos a seguinte sintaxe:

(valor ou expressão) + (comparador) + (segundo valor ou expressão)

Observe a tabela abaixo:

Comparadores	Descrição
==	Igual. Resulta em TRUE se as expressões forem iguais.
===	Idêntico. Resulta em TRUE se as iguais e do mesmo tipo de dados.
!=	Diferente. Resulta verdadeiro se as variáveis foram diferentes.
<	Menor ou menor que. Resulta TRUE se a primeira expressão for menor.
>	Maior ou maior que. Resulta TRUE se a primeira expressão for maior.
<=	Menor ou igual. Resulta TRUE se a primeira expressão for menor ou igual.
>=	Maior ou igual. Resulta TRUE se a primeira expressão for maior ou igual.

Veja um exemplo prático:

a <= b

Compara se a é menor ou igual a b, onde, retorna verdadeiro (TRUE), caso contrário retorna falso (FALSE).

Para testarmos essas comparações podemos utilizar o condicional “?:” (ou **ternário**), sua sintaxe é a seguinte:

(expressão booleana) ? (executa caso verdadeiro) : (executa caso falso);

Agora podemos ver um exemplo envolvendo as sintaxes e empregabilidade dos comparadores:

```
<script type="text/javascript">
    var a = 15;
    var b = "42";
    var c = 42.0;

    document.writeln(b ==  c ? "verdadeiro" : "falso"); // VERDADEIRO
    document.writeln(b === c ? "verdadeiro" : "falso"); // FALSO
    document.writeln(b !=  c ? "verdadeiro" : "falso"); // FALSO
    document.writeln(a < c ? "verdadeiro" : "falso");   // VERDADEIRO
    document.writeln(a > c ? "verdadeiro" : "falso");   // FALSO
    document.writeln(a <= c ? "verdadeiro" : "falso");  // VERDADEIRO
    document.writeln(a >= c ? "verdadeiro" : "falso");  // FALSO
</script>
```

Nesse exemplo declaramos e iniciamos três variáveis. Usamos então o comando **document.write()** para imprimir o resultado, onde o condicional "**?:"** foi utilizado. Iniciamos as comparações de a, b e c, caso a comparação individual retorne TRUE, imprime verdadeiro, caso retorne FALSE, imprime falso. Observe que o comparador "===" compara o valor e o tipo, retornando FALSE por b se tratar de um tipo inteiro, e c um tipo ponto flutuante, já o comparador "==" compara somente os valores onde 45 é igual a 45.0 retornando verdadeiro. Também podemos usar o operador "**!=="** onde tem a função semelhantemente ao operador "!=" , mas retorna TRUE se os tipos forem diferentes. Se a variável for do tipo booleano, podemos compará-los assim:

a == TRUE, a == FALSE

3.5 Operadores lógicos ou booleanos

São utilizados para avaliar expressões lógicas. Estes operadores servem para avaliar expressões que resultam em valores lógico sendo verdadeiro ou falso:

E	&&
OU	‖
XOR	^
Não	!

Esses operadores tem a finalidade de novas proposições lógicas composta a partir de outras proposições lógicas simples. Observe:

Operador	Função
não	negação
e	conjunção
Ou exclusivo(xor)	disjunção exclusiva
ou	disjunção

Com isso podemos construir a Tabela verdade onde trabalhamos com todas as possibilidades combinatória entre os valores de diversas variáveis envolvidas, as quais se encontram em apenas duas situações (V e F), e um conjunto de operadores lógicos. Veja o comportamento dessas variáveis:

3.6 Operação de Negação:

A	(!) não A

F	V
V	F

Trazendo para o nosso cotidiano:

Considerando que A = “Está chovendo”, sua negação seria : “Não está chovendo”. Considerando que A = “Não está chovendo” sua negação seria : “Está chovendo”

3.7 Operação de conjunção:

A	B	A e B
F	F	F
F	V	F
V	F	F
V	V	V

Trazendo para o nosso cotidiano:

Imagine que acontecerá um casamento:

Considerando que A = “Noivo presente” e B = “Noiva presente”.

Sabemos que um casamento só pode se realizar, se os 2 estejam presente

3.8 Operação de disjunção

- **Não exclusiva**

A	B	A ou B
F	F	F
F	V	V
V	F	V
V	V	V

Trazendo para o nosso cotidiano:

Imagine que acontecerá uma prova e para realizá-la você precisará da sua Identidade ou título de eleitor no dia da prova.

Considerando que A = “Identidade” e B = “Título de eleitor”.

Sabemos que o candidato precisa de pelo menos 1 dos documentos para realizar a prova.

- **Exclusiva**

A	B	A xor B
F	F	F
F	V	V
V	F	V
V	V	F

Trazendo para o nosso cotidiano:

Imagine que Cateriny nasceu em Fortaleza. Se for indagado para Cateriny se ela nasceu em Crato, Juazeiro do Norte, Sobral ou Fortaleza. Ela iria escolher apenas umas dessas opções que seria Fortaleza. Não há possibilidade de Cateriny nascer em mais de uma cidade.

Considerando que A = “Crato” e B = “Juazeiro do Norte” e C=”Sobral” e D=”Fortaleza”.

A expressão: A xor B xor C xor D só poderá ser verdadeira se tiver apenas uma reposta válida.

São chamados de operadores lógicos ou booleanos por se tratar de comparadores de duas ou mais expressões lógicas entre si, fazendo agrupamento de testes condicionais e tem como retorno um resultado booleano.

Na tabela abaixo temos os operadores e suas descrições:

Operador	Descrição
(a && b)	E : Verdadeiro se tanto a quanto b forem verdadeiros.
(a || b)	OU : Verdadeiro se a ou b forem verdadeiros.
(a^b)	XOR: somente um pode ser verdade para que a expressão seja verdadeira(bit-a-bit)
(! a)	NOT : Verdadeiro se a for falso, usado para inverter o resultado da condição.

Exemplo:

```
<script type="text/javascript">
    var a = 10 > 2;   // 'a' recebe TRUE
    var b = 12 >= 12; // 'b' recebe TRUE
    var c = false;    // 'c' recebe FALSE

    document.writeln(b && a ? "SIM" : "NÃO"); // SIM
    document.writeln(b || c ? "SIM" : "NÃO"); // SIM
    document.writeln(b ^ a ?  "SIM" : "NÃO"); // NÃO
    document.writeln((c!) ?   "SIM" : "NÃO"); // SIM
</script>
```

3.9 Precedência de Operadores

Agora já conhecemos uma boa quantidade de operadores no JavaScript, falta agora conhecer a precedência de cada um deles, ou seja, quem é mais importante, qual operador é avaliado primeiro e qual

```
<script type="text/javascript">
   document.write(5+2*6); // Resultado: 17
 </script>
```

é avaliado em seguida. Observe o seguinte exemplo:

O resultado será 17, pois o operador * tem maior precedência em relação ao operador +. Primeiro ocorre a multiplicação 2*6, resultando em 12, em seguida a soma de 5 + 12. Caso desejar realizar a operação com o operador + para só em seguida realizar a operação com o operador *, temos que fazer conforme o exemplo abaixo:

```
<script type="text/javascript">
   document.write((5+2)*6); // Resultado: 42
 </script>
```

Observe que utilizamos os parênteses para determinarmos quem deve ser executado primeiro, assim alterando o resultado para 42. Os parênteses determinam qual bloco de código executa primeiro, e também serve para isolar determinadas operações. Veja mais um exemplo onde as operações são feitas separadamente. Primeiro executa a soma, em seguida a subtração e só então é executado a multiplicação, imprimindo um resultado final **21**:

```
<script type="text/javascript">
   document.write((5+2)* (6-3)); // Resultado: 21
 </script>
```

Exemplo:

A tabela a seguir lista os operadores JavaScript, ordenados da maior até a menor precedência. Operadores com a mesma precedência são avaliados da esquerda para a direita.

operador	Descrição
. [] ()	Acesso a campos, indexação de matriz, chamadas de função e agrupamento de expressões
++ -- - ~ ! delete new typeof void	Operadores unários, tipo de dados de retorno, criação de objetos, valores indefinidos
* / %	Multiplicação, divisão, divisão de módulo
+ - +	Adição, subtração, concatenação de cadeias de caracteres
<< >> >>>	Deslocamento de bits
< <= > >= instanceof	Menor que, menor que ou igual a, maior que, maior que ou igual a, instância de
== != === !==	Igualdade, desigualdade, igualdade estrita e desigualdade estrita
&	AND bit a bit
^	XOR bit a bit
|	OR bit a bit
&&	Lógico AND
||	OU lógico
?:	Condicional
= OP=	Atribuição, atribuição com operação (como += e &=)
,	Avaliação múltipla

É importante lembrar que primeiro o JavaScript executará todas as operações que estiverem entre parênteses, se dentro dos parênteses houver diversas operações, a precedência dos operadores será utilizada para definir a ordem. Após resolver todas as operações dos parentes, o JavaScript volta a resolver o que está fora dos parênteses baseando-se na tabela de precedência de operadores. Havendo operadores de mesma prioridade o JavaScript resolverá a operação da esquerda para direita.

Também podemos trabalhar com procedência de parênteses, fazendo associações com um ou mais operadores, observe o seguinte exemplo:

```
<script type="text/javascript">
   document.write( (3+2)*(9-3) / ( 16-((5+2)*2) ) );
   // Resultado: 15
</script>
```

Seguindo a ordem de procedência temos:

(5) * (6) / (16 - ((7)*2)) >>> 5 * 6 / (16 - (14)) >>> 5 * 6 / 2 >>> 30 / 2

Resultado: 15

Observe que primeiro executa todos os parênteses, e só então temos as procedências das demais operações.

Exercícios Propostos

Qual a finalidade dos operadores de strings?

Quais os operadores de decremento e incremento? Cite alguns exemplos:

Qual a finalidade do operador aritmético %(modulo)?

Cite os operadores relacionais, mostre alguns exemplos.

Quais operadores lógicos ou booleanos?

Quais os operadores de atribuição?

Qual a sintaxe do uso de ternário e cite um exemplo?

Observe o código abaixo e diga quais das operações são executadas primeiro, coloque a resposta em ordem decrescente.

A = 8*5-3+4/2+19%5/2+1;

Faça testes com os operadores relacionais substituindo o operados > do código fonte abaixo:

```
<script>
var1 = 2.2564;
var2 = 2.2635;
document.write(var1 > var2 ? “sim” : “não”);
</script>
```

2. Usando o operador de concatenação “+” para montar a seguinte frase abaixo:

```
<script>
a = “de”; b = “é um”; c = “comunicação”; c = “a”;
d = “internet”; e = “meio”;
document.write( ..... );
</script>
```

4.0 Caixa de Diálogo

Um recurso interessante de JavaScript é a possibilidade de criar caixas de diálogo simples, que podem ser muito informativas aos usuários que a visualizam.

Essas caixas de diálogo podem ser de alerta, de confirmação ou de prompt de entrada. Todas elas são chamadas de forma simples e intuitiva por uma função.

4.1 Alert

As caixas de diálogo de alerta são simples e informativas. Elas, geralmente, são utilizadas em validação de formulários ou bloqueio de ações.

Sua função é mostrar apenas uma mensagem com um botão de confirmação para que esta seja fechada.

Para chamar esta caixa de diálogo usamos a função **alert()**. Esta função recebe como parâmetro uma string que será a mensagem a ser exibida. Vejamos o código abaixo:

```
<script type="text/javascript">
    alert ("Esta é uma caixa de diálogo ALERT do javascript!")
</script>
```

Em caixas de diálogo há a possibilidade de controlar o fluxo de texto usando `\n` para a quebra de linhas.

```
<script type="text/javascript">
    alert ("javascript \n Caixa de dialogo \n Alert")
</script>
```

4.2 Prompt

A caixa de diálogo de prompt nos possibilita requerer uma entrada ao usuário apesar de não ser tão útil, pois esse recurso pode facilmente ser substituído por um campo de texto feito em HTML.

Para chamarmos esta caixa de diálogo, usamos a função `prompt()` que recebe uma string como parâmetro. Esse parâmetro será a mensagem a ser exibida dentro da caixa de diálogo.

A caixa de diálogo de prompt possui três elementos: um campo input para texto, um botão OK e outro CANCELAR.

A função `prompt ()` sempre irá retornar um valor, ou seja, podemos gravar o resultado da função em uma variável ou algo assim. Se clicarmos no botão OK, o valor a ser retornado será o que estiver escrito no campo de texto, mesmo se ele estiver vazio. Se clicarmos em CANCELAR, o valor retornado será **null**.

Abaixo criamos um exemplo no qual exige que o usuário digite o nome dele. Para isso, colocamos o prompt dentro de uma estrutura de repetição *while* que tem a seguinte condição: se o resultado for null (ou seja, se o usuário clicar em cancelar), ou então, se o resultado for vazio (ou seja, se o usuário não digitar nada e clicar no OK), neste caso, deve-se executar a repetição.

Dessa forma nos asseguramos que o usuário sempre irá digitar alguma coisa dentro da caixa de diálogo.

```
<script type="text/javascript">
    var nome;
    do {
        nome = prompt ("Qual é o seu nome?");
    }while (nome == null || nome == "");
    alert ("Seu nome é: "+nome);
</script>
```

4.3 Confirm

A caixa de diálogo de confirmação é chamada pela função `confirm()` e tem apenas dois botões: um OK e outro CANCELAR. Assim como a função prompt(), a função confirm() também retorna um valor que pode ser true (verdadeiro) ou false (falso).

Como confirm() retorna um valor booleano, isso o torna ideal para ser usado com uma estrutura seletiva *if*. Por exemplo, podemos usar a caixa de diálogo de confirmação antes de redirecionarmos uma página para executar uma rotina para apagar algum registro do banco de dados.

No exemplo abaixo, não iremos tão profundamente quanto o cenário acima, pois envolve mais do que simples JavaScript. Aqui, apenas iremos demonstrar o resultado do clique em algum dos dois botões.

```
<script type="text/javascript">
    decisao = confirm("Clique em um botão!");
    if (decisao) {
        alert ("Você clicou no botão OK, \n"+
               "porque foi retornado o valor: "+decisao);
    } else{
        alert ("Você clicou no botão CANCELAR, \n"+
               "porque foi retornado o valor: "+decisao);
    };
</script>
```

Exercícios Propostos

Usando a instrução alert, faça um algoritmo que exibe para o usuário a mensagem: “Bom dia usuário.”.

Faça um algoritmo que solicite ao usuário que digite seu nome, após o usuário dar entrada da informação exiba a seguinte mensagem para o mesmo: “Bom dia nome_usuario.”. Onde há nome_usuario será

colocado o nome que o usuário digitou.

Faça um algoritmo que solicite o ano que o usuário nasceu e depois o ano atual, seu código irá pegar essas informações e calcular a idade do usuário e exibi-la na tela usando alert().

Faça um algoritmo que receba dois números inteiros, digitados pelo usuário, e exiba em um único alert() o resultado da soma, subtração, multiplicação e divisão entre os dois números digitados.

Faça um algoritmo que peça sua amado(a) em casamento ou namoro, utilize a instrução confirm(). Caso ele(a) aceite exiba a mensagem “Eu aceito”, caso contrário exiba a mensagem “Desculpa, não posso aceitar”.

Faça um algoritmo de validação de login. O usuário irá digitar o login e depois a senha. Se o login digitado é igual a “root” e a senha “qwe123”, então exiba a mensagem “Acesso permitido”, senão exiba “Login/Senha inválidos”;

Escreva um algoritmo que calcula quantos meses um funcionário trabalhou para uma empresa. Solicite o dia, mês e ano que um funcionário foi contratado e depois o dia, mês e ano que ele foi demitido. Calcule quantos meses esse funcionário trabalhou nessa empresa.

Escreva um algoritmo que receba a idade do usuário e se a idade for maior o u igual a 18 exiba a mensagem “Sou um adulto”, senão exiba a mensagem “Sou uma Criança/Adolescente”.

Escreva um algoritmo que solicite ao usuário uma lista de compras para se fazer no supermercado. Essa lista terá 10 produtos e após todos serem digitados exiba a lista em um só alert() utilizando essa formatação:

1 – arroz

2 – feijão

3 – macarrão

4 – farofa

Escreva um algoritmo que receba 3 valores inteiros que representam a tamanho dos lados de um triângulo. Usando o operador ternário exiba a mensagem “Sou equilátero” se todos os lados forem iguais, “Sou isósceles” se dois lados forem iguais e “Sou escaleno “, se todos os lados forem diferentes.

5.0 Estruturas de controle e repetição

Objetivos

Mostra estruturas de controle e sua aplicação prática em JavaScript; Definir qual a principal finalidade dessas estruturas; Mostrar exemplos em sua sintaxe;.

As estruturas que veremos a seguir são comuns para as linguagens de programação imperativas, bastando descrever a sintaxe de cada uma delas resumindo o funcionamento. Independente do JavaScript, boa parte das outras linguagens de programação tem estruturas bem semelhantes, mudando apenas algumas sintaxes.

5.1 Blocos de controle

Um bloco consiste de vários comandos agrupados com o objetivo de relacioná-los com determinado comando ou função. Em comandos como if, for, while, switch e em declarações de funções blocos podem ser utilizados para permitir que um comando faça parte do contexto desejado. Blocos em JavaScript são delimitados pelos caracteres "{" e "}". A utilização dos delimitadores de bloco em uma parte qualquer do código não relacionada com os comandos citados ou funções não produzirá efeito algum, e será tratada normalmente pelo interpretador. Outro detalhe importante: usar as estruturas de controle sem blocos delimitadores faz com que somente o próximo comando venha ter ligação com a estrutura. Observe os exemplos:

```
<script type="text/javascript">
    var a = 0;
    if (a>2)
        alert("comando 1");
        alert("comando 2");
</script>
```

Observe que temos um comando IF, onde é passado a ele uma expressão booleana que retorna verdadeiro ou falso.

O resultado da expressão é FALSE (falso), pois 0 não é maior que 2, fazendo com que o IF não execute o **echo** com "comando1". Somente o segundo **echo** é executado, pois não pertence ao IF declarado.

Mas se quisermos que mais de um comando pertença a estrutura de controle, será usado blocos de comandos ({comando;}), onde através deles podemos delimitar e organizar os códigos.

```
<script type="text/javascript">
    var a = 0;
    if (a>2) {
        alert("comando 1");
        alert("comando 2");
    }
</script>
```

No código ao lado, temos um bloco onde inserimos dois comandos. Observe que eles não serão executados, pois a expressão booleana passada para o IF é falsa.

5.2 IF e ELSE

Essa estrutura condicional está entre as mais usadas na programação. Sua finalidade é induzir um desvio condicional, ou seja, um desvio na execução natural do programa. Caso a condição dada pela expressão seja satisfeita, então serão executadas a instruções do bloco de comando. Caso a condição não seja satisfeita, o bloco de comando será simplesmente ignorado. Em lógica de programação é o que usamos como "SE (expressão) ENTÃO {comando:}".

–

```
if (expressão)
    comando;
if
(expressão){
    comando1;
    comando2
```

Sintaxe:

```
<script type="text/javascript">
    var idade = 16;
    if (idade >18) {
        var texto = "Maior idade";
    };
    alert(texto);
</script>
```

Exemplo:

Caso a condição não seja satisfatória (FALSE), podemos atribuir outro comando pertencente ao IF chamado ELSE, como se fosse a estrutura SENÃO em lógica de programação.

Sintaxe:

```
if
(expressão)
    comando;
eles

    comando

; if

(expressão)

{
    comand
    o1;
```

Exemplo:

```
<script type="text/javascript">
    var idade = 16;
    if (idade>18) {
        var texto = "Maior de idade";
    }else{
        var texto = "Menor de idade";
    }
    alert(texto);
</script>
```

Nesse exemplo temos uma expressão booleana onde retorna falso, com isso o IF não executa, passando a execução para o eles, que por sua vez executa e atribuí o valor “menor idade” a variável texto.

Em determinadas situações é necessário fazer mais de um teste, e executar condicionalmente diversos comandos ou blocos de comandos. Isso é o que podemos chamar de “If's encadeados”, onde usamos a estrutura **ELSE IF** .Para facilitar o entendimento de uma estrutura do tipo:

```
if (expressao1)
    comando1;

        else
    if (expressao2)

         comando2;
    else

         if (expressao3)
               comando3;
         else comando4;
```

Sintaxe:

```
<script type="text/javascript">
    var idade = 16;
    if (idade>18) {
        var texto = "Maior de idade";
    }else if (idade < 12){
        var texto = "Criança";
    }else{
        var texto = "Adolescente";
    }
    alert(texto);
</script>
```

Exemplo:

Exercício rápido:

1°) Faça uma script em JavaScript que possua 4 notas de um aluno (cada uma em uma variável). Depois calcule e imprima a média aritmética das notas e a mensagem de aprovado para média superior ou igual a 7.0 ou a mensagem de reprovado para média inferior a 7.0.

2°) Faça um script em JavaScript que receba a idade de um nadador (representada por uma variável chamada “idade”) e imprima a sua categoria seguindo as regras:

Categoria	**Idade**
Infantil A	5 - 7 anos

Infantil B	8 - 10 anos
Juvenil A	11- 13 anos
Juvenil B	14- 17 anos

5.3 Atribuição condicional (ternário)

Como já vimos exemplos de atribuição condicionais (ternários), podemos defini-los usando a sintaxe: (Expressão booleana) ? (executa caso verdadeiro) : (executa caso falso);

Isso se aplica quando queremos uma estrutura resumida, onde podemos ter um resultado mais direto, como por exemplo, atribuir um valor a uma variável dependendo de uma expressão. Observe o exemplo abaixo onde envolvemos uma variável do tipo string, porém o valor atribuído a essa variável deverá ser de acordo com o valor da idade:

```
<script type="text/javascript">
    var idade = 16;
    var texto = idade > 18 ? "Maior de idade" :"Menor de idade";
    alert(texto);
</script>
```

É uma estrutura parecida com IF e ELSE, onde dependendo da expressão booleana podemos executar um bloco ou não.

Exercício rápido:

1°) Faça uma script em JavaScript que receba um número representado por uma variável. Verifique se este número é par ou ímpar e imprima a mensagem.
2°) Crie outro script baseando em um seguro de vida com as seguintes regras:

Idade:	**Grupo de Risco :**
18 a 24	- Baixo
25 a 40	- Médio
41 a70	- Alto

5.4 SWITCH

Observe que quando temos muitos “if's encadeados” estamos criando uma estrutura que não é considerada uma boa prática de programação. Para resolver esse problema temos uma estrutura onde sua funcionalidade é semelhante ao **ELSE IF.** O comando SWITCH é uma estrutura que simula uma bateria de teste sobre uma variável. Frequentemente é **necessário** comparar a mesma variável com valores diferentes e executar uma ação especifica em cada um desses valores.

```
switch(expressão){

    case "valor
      1":
      comandos;

    case "valor
      1":
      comandos;
    case "valor
      1":
      comandos;
```

Sintaxe:

```
<script type="text/javascript">
    var numero = 2;
    switch(numero){
        case 1:
            alert("opção 1");
        case 2:
            alert("opção 2");
        case 3:
            alert("opção 3");
        case 4:
            alert("opção 4");
        case 5:
            alert("opção 5");
    }
</script>
```

Exemplo:

Resultado: opção 2: opção 3:opção 4:opção 5:

Nesse exemplo temos o número = 2, onde o switch compara com os case's o valor recebido, o bloco que é executado é do segundo case, porém, os demais também são executados para que tenhamos um resultado satisfatório temos que usar em cada case um comando chamado **break**. No qual tem a função de para o bloco de execução.

5.5 SWITCH com BREAK

Break é uma instrução (comando) passada quando queremos parar o fluxo da execução de um programa. Em JavaScript, ele tem a mesma função que é "abortar" o bloco de código correspondente.

Observe o mesmo exemplo com o uso de break:

Temos agora como resultado “opção 2:”. O comando break fez com que os demais case's abaixo do 'case 2' não sejam executados.

```
<script type="text/javascript">
    var numero = 2;
    switch(numero){
        case 1:
            alert("opção 1");
            break;
        case 2:
            alert("opção 2");
            break;
        case 3:
            alert("opção 3");
            break;
        case 4:
            alert("opção 4");
            break;
        case 5:
            alert("opção 5");
            break;
    }
</script>
```

Obs.: Além de números podemos também comparar outros tipos como string, pontos flutuantes e inteiros, veja um exemplo abaixo:

```
<script type="text/javascript">
    var numero = "opc 2";
    switch(numero){
        case "opc 1":
            alert("opção 1");
            break;
        case "opc 2":
            alert("opção 2");
            break;
    }
</script>
```

Mas o que acontece se não tivermos um valor que seja satisfatório aos casos existentes no switch? A resposta é bem simples, nenhum dos blocos seriam executados, porém temos um comando onde determinamos uma opção padrão caso nenhuma das outras venha ter resultado que satisfaça a expressão passada para o switch chamada default (padrão).

```
<script type="text/javascript">
    var numero = "opc 3";
    switch(numero){
        case "opc 1":
            alert("opção 1");
            break;
        case "opc 2":
            alert("opção 2");
            break;
        default:
            alert("opção inválida");
    }
</script>
```

Veja um exemplo:

Resultado: opção inválida

A instrução passada não condiz com nenhum dos casos existentes. Por esse motivo o bloco pertencente ao comando default será executado.

O comando default pode ser inserido em qualquer lugar dentro do switch, porém caso isso aconteça, o uso do comando break deve ser adicionado para evitar que os case's abaixo

A partir de agora trabalharemos as estruturas de repetição. Elas muito utilizadas nas linguagens de programação.

1°) Faça um script em JavaScript usando switch, onde receba uma variável e mostre as seguintes opções: 1 - módulo.
2 - somar.
3 - subtrair.
4 - multiplicar.

5.6 WHILE

O WHILE é uma estrutura de controle similar ao IF, onde possui uma condição para executar um bloco de comandos. A diferença primordial é que o WHILE estabelece um laço de repetição, ou seja, o bloco de comandos será executado repetitivamente enquanto a condição passada for verdadeira. Esse comando pode ser interpretado como "ENQUANTO (expressão) FAÇA { comandos...}".

Sintaxe:

```
while
    (expressão){
    comandos;
```

Quando estamos usando um laço de repetição, podemos determinar quantas vezes ele deve ou não se repetir. Para trabalharmos com essa contagem de quantas vezes o laço deve se repetir, usaremos incremento ou decremento de uma variável conforme vimos no capitulo de operadores em JavaScript. Observe o exemplo abaixo:

```
<script type="text/javascript">
    var a = 1;
    while(a < 10){
        alert(a);
        a++;  //incrementa a variável
    }
</script>
```

Sequência do resultado em caixa de diálogo: 1 2 3 4 5 6 7 8 9

Nesse exemplo criamos um laço de repetição que tem como condição a < 10, a cada laço é executado um incremento na variável a, fazendo com que o seu valor aumente até a condição não ser mais satisfatória.

Dicas: Tenha cuidado quando estiver trabalhando com loop's (laço de repetição), pois caso a expressão passada esteja errada, pode ocasionar em um loop infinito fazendo com que o bloco de código se repita infinitamente. Isso pode ocasionar um travamento do navegador ou até mesmo do próprio servidor WEB.

Vamos ver agora um exemplo em que o laço se repete de forma automática, onde quem determina o loop são propriedades do JavaScript e não um número determinado pelo programador.

A propriedade length é chamada logo após uma variável de texto e retorna a quantidade de caracteres incluindo também os espaços em branco. Ele poderia ser aplicado diretamente no alert, mas no exemplo, ele determina a quantidade de loop's.

```
<script type="text/javascript">
    var a = 1;
    var texto = "Projeto e-Jovem";
    //length conta o tamanho da string
    while(a < texto.length){
        a++;  //incrementa a variável
    }
    alert("O texto possui "+a+" caracteres");
</script>
```

Resultado: a frase possui 15 caracteres

Exercício rápido:

1°) Faça um script que conte de 1 até 100.
2°) Faça um script que imprima na tela números de 3 em 3 iniciando com 0 até 90, ex: 0,3,6,9...

5.7 DO...WHILE

O laço do...while funciona de maneira bastante semelhante ao while, com a simples diferença que a expressão é testada ao final do bloco de comandos. O laço do...while possui apenas uma sintaxe que é a seguinte:

Sintaxe:

```
do {
    comando
    ;
. . .
    comando;
} while
```

Exemplo:

```
<script type="text/javascript">
    var a = 1;
    do{
        alert(a);
        a++;
    }while(a<10);
</script>
```

Dicas: Talvez na criação de alguma página ou sistema web, seja necessário executar um bloco de código existente em um laço de repetição pelo menos uma vez, nesse caso podemos usar o do...while.

Sequência do resultado em caixa de diálogo: 1 2 3 4 5 6 7 8 9

Exercício rápido:

1°) Faça um script que conte de -1 até -100 usando “do while”.

2°) Faça um script que imprima na tela somente números pares de 2 até 20 com do while.

5.8 FOR

Outra estrutura semelhante ao while é o for, onde tem a finalidade de estabelecer um laço de repetição em um contador. Sua estrutura é controlada por um bloco de três comandos que estabelecem uma contagem, ou seja, o bloco de comandos será executado determinado número de vezes.

Sintaxe:

```
for( inicialização; condição;
incremento ){ comandos;
}
```

Parâmetros	Descrição
Inicialização	Parte do for que é executado somente uma vez, usado para inicializar uma variável.
Condição	Parte do for onde é declarada uma expressão booleana.
Incremento	Parte do for que é executado a cada interação do laço.

Lembrando que o loop do for é executado enquanto a condição retornar expressão booleana verdadeira. Outro detalhe importante é que podermos executar o incremento a cada laço, onde possibilitamos adicionar uma variável ou mais. Mostraremos agora um exemplo fazendo um comparativo entre a estrutura de repetição do while e também a do for de forma prática.

Exemplo:

```
<script type="text/javascript">
// COM WHILE
    var a = 1;
    while(a<10){
        alert(a);
        a++;
    }
</script>
```

WHILE

```
<script type="text/javascript">
// COM FOR
    for(a=1;a<10;a++){
        alert(a);
    }
</script>
```

FOR

*Ambos exemplos geram a mesma sequência em caixa de diálogo com o resultado:*1 2 3 4 5 6 7 8 9

O **for** não precisa ter necessariamente todas as expressões na sua estrutura, com isso podemos criar um

exemplo de **for** onde suas expressões são declaradas externamente.

```
<script type="text/javascript">
    a = 1;
    for(;a<10;){
        alert(a);
        a++;
    }
</script>
```

Observe nesse exemplo uma proximidade muito grande do comando while. Apesar de ser funcional, não é uma boa prática de programação utilizar desta forma.

A estrutura **“while”** e **“do while”** normalmente é usado quando o programador não sabe quantos loops o sistema irá realizar, já o **“for”** informamos explicitamente qual será o ponto de partida, o seu limite, e valor de incremento ou decremento.

Exercício rápido:

1°) Faça um script que receba duas variáveis “a” e “b”, logo após imprima os números de intervalos entre eles com o uso de “for”.ex: a=5 ,b = 11, imprime : 5,6,7,8,9,10,11.

5.9 FOREACH

O **foreach** é um laço de repetição para interação em array's ou matrizes, o qual estudaremos com mais detalhes no próximo capítulo. Trata-se de um **for** mais simplificado que compõe um vetor ou matriz em cada um de seus elementos por meio de sua cláusula **IN.**

```
 vetor = new Array();
foreach(incide in vetor){
  comandos;
}
```

Sintaxe:

```
<script type="text/javascript">
    var texto = new Array("HTML","CSS","javascript");

    for (var indice in texto) {
        alert(texto[indice]);
    };
</script>
```

Exemplo:

Resultado: *HTML CSS JAVASRIPT*

Veremos adiante que um array é uma variável composta por vários elementos. No caso do exemplo anterior, esses elementos são nomes de cursos. A finalidade do foreach é justamente a cada laço, pegar cada índice desses valores e atribuir a uma variável, até que tenha percorrido todo array e assim, finalizar o laço. Podemos saber em qual posição o elemento se encontra no array, para isso basta imprimir o índice.

Observe o exemplo:

```
<script type="text/javascript">
    var texto = new Array("HTML","CSS","javascript");

    for (var indice in texto) {
        alert(indice+" - "+texto[indice]);
    };
</script>
```

***Resultado**:*

0-HTML

1-CSS

2-JAVASCRIPT

Nesse exemplo observamos que cada elemento do array possui um índice (chave), imprimindo na tela o número da posição e o valor guardado.

5.10 BREAK.

Outro comando importante é o break, usado para abortar (parar) qualquer execução de comandos como SWITCH, WHILE, FOR, FOREACH, ou qualquer outra estrutura de controle. Ao encontrar um break dentro de um desses laços, o interpretador JavaScript interrompe imediatamente a execução do laço, seguindo normalmente o fluxo do script.

Sintaxe:

```
while....
   for....
      break <quantidades de
      níveis>;
```

Vamos ver um exemplo com o uso de **break** dentro de um laço de repetição (no caso o **for**), onde criamos

```
<script type="text/javascript">
    var a = 1;
    for ( ; ; ) { //Loop Infinito
        document.write("<p>"+a+"</p>");
        if (a===10) {
            break;   //Para o laço
        };
        a++;  // Incremento
    };
</script>
```

um laço infinito, porém colocamos um if com a condição de parar o laço através do **break**. Observe:

Podemos notar nesse exemplo a criação de um laço(loop) infinito, que ocorre quando tiramos a condição

do **for**, ou atribuímos “for(; true ;)”, porém a condição fica na responsabilidade do if, quando o valor de a e igual a 10, faz com que o if execute o **break**, fazendo com que o laço pare de funcionar.

5.11 CONTINUE

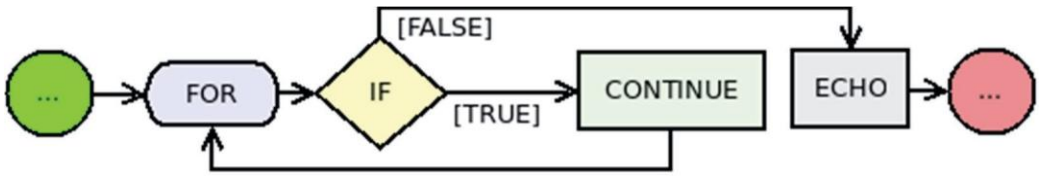
A instrução **continue**, quando executada em um bloco de comandos for/while, ignora as instruções restantes até o fechamento em “}”. Dessa forma, o programa segue para a próxima verificação da condição de entrada do laço de repetição, funciona de maneira semelhante ao break, com a diferença que o fluxo ao invés de sair do laço volta para o início dele. Veja um exemplo:

```
<script type="text/javascript">
    for ( var a = 0; a<20 ;a++) {
        if( a % 2){
            continue;
        }
        document.write(+a+" ");
        };
</script>
```

***Resultado**:* 0,2,4,6,8,10,12,14,16,18

Podemos observar a seguinte lógica no exemplo acima:

Criamos um laço que tem 20 interações de repetição. Logo após temos um if, onde, quando o resto da divisão por 2 for igual a 0 (número par), o valor booleano será “false”. Quando não for igual a 0, significa que a variável a é um número ímpar (ex: 5%2 = 1), então temos um valor booleano “true”. Isso significa que o if executa somente quando os números forem ímpares. Adicionamos um continue, que ao executar o if, faz com que volte novamente para o início do for impedindo de alcançar o echo em seguida. Com isso, em vez de mostramos os números impares, imprimimos somente os números pares incluindo o 0. Resumimos que o código só passa adiante quando o if não executa o continue. Fluxograma:



Exercícios Propostos

Qual a principal finalidade de uma estrutura de controle?

Qual a principal finalidade de uma estrutura de repetição?

Crie um código com a uma condição ternária onde receba um valor booleano e de acordo com o valor passado na expressão, deve imprimir “sim” ou “não”.

Com o comando IF e ELSE crie um código que determine se uma expressão é verdadeira ou

falsa.

Qual a finalidade da estrutura de controle SWITCH e cite um exemplo onde comparamos uma opção com 4 casos diferente?

Crie um contador de 1 até 20 usando a estrutura de repetição WHILE.

Crie um contador de 1 até 100 usando DO WHILE. EP06.8: Crie um contador de 100 até 1 usando FOR.

Qual a fonalidade de um FOREACH?

Crie um código onde podemos para a execução de um laço infinito com o uso de BREAK.

Como podemos determinar o uso de CONTINUE e qual a sua aplicação prática em JavaScript.

Crie um código com as seguintes características:

Deverá receber um valor inicial e outro final (crie duas variáveis para esse fim).

Como o comado FOR crie um laço onde a contagem e determinada pelo valor inicial e final?

Dentro do for deverá conter um IF e ELSE responsável por compara os valores passado a ele e imprimir os pares e impares. Exemplo:

IF(valor%2==0)

echo valor. "é um número par"; ELSE

echo valor. "é um número ímpar";

Exemplo prático: foi passado o número inicial 8 e o final 15, então o script JavaScpit deverá imprimir o intervalo entre esse número ou seja 8,9,10,11,12,13,14,15, mostrando quais deles são pares e quais são ímpares.

6.0 Objetos em JavaScript

Quase “Tudo” em JavaScript é um objeto: uma String, um número, uma matriz, uma data, um botão ....

Em JavaScript, um objeto, são variáveis que guardam atributos e métodos. Atributos são características de um objeto.

Métodos são ações que um objeto pode realizar;

Vamos exemplificar com um objeto da vida real: Objeto Carro.

carro.

As propriedades do carro incluem a montadora, o modelo, o peso, a cor, etc...

Todos os carros têm essas propriedades, mas os valores dessas propriedades diferem de carro para

Os métodos do carro são ligar(), acelerar(), freiar(), etc

Todos estes métodos pertencem ao objeto carro, mas eles são realizados em momentos diferentes.

6.1 Objeto String, Acessando atributos e métodos

Em JavaScript como já mencionado quase “tudo” é objeto. Quando declaramos uma variável dessa forma:

```
var txt = "Hello";
```

O que realmente estamos fazendo é criando um objeto String JavaScript. O objeto String tem vários métodos e alguns atributos já prontos para facilitar o seu trabalho quando for preciso manipular Strings, os quais são chamados built-in, ou seja, métodos que são nativos do JavaScript.

6.2 Acessando atributos

Para acessarmos um atributo ou métodos de um objeto usamos o operador ponto (.), logo após o nome do objeto.

Por exemplo, vamos acessar uma propriedade do Objeto String txt, criado anteriormente, chamado de **lenght**:

```
var txt = "Hello";
var tamanhoDaString = txt.length;
```

Se executarmos a instrução document.write(tamanhoDaString); será mostrado o valor 5 na página.

6.3 Acessando Métodos

Para acessarmos os métodos de um objeto também utilizamos o operador ponto (.), logo após o nome do objeto, a diferença é que após o nome do método que será chamado digitamos parênteses ( ), nos quais podemos colocar ou não argumentos dentro.

Abaixo a sintaxe padrão para acesso de métodos de um objeto:

```
nomeObjeto.nomeMetodo(<argumento1,argumento2,argumentoN>)
```

Legenda:

- <argumento1,argumento2,argumentoN> ‹ dependendo do método que você deseja chamar esse parâmetro pode existir ou não;
- nomeObjeto ‹ nome do objeto JavaScript;
- nomeMetodo ‹ nome do método que você deseja chamar/executar;

Neste exemplo vamos utilizar um dos métodos nativos do objeto String JavaScript, o método **toUpperCase()**.

```
<script type="text/javascript">
    var message = "Hello World!";
    var x = message.toUpperCase();
    document.write(x);
</script>
```

Este método é responsável por converter todos os caracteres de uma String para maiúsculo. A saída desse trecho de código JavaScript é essa:

HELLO WORLD!

Abaixo uma lista com todos os métodos e atributos que um objeto String JavaScript possui:

Atributos:

length	***prototype***	***constructor***

Métodos:

charAt()	***charCodeAt()***	***concat()***	***fromCharCode()***
indexOf()	***lastIndexOf()***	***match()***	***replace()***
search()	***slice()***	***split()***	***substr()***
substring()	***toLowerCase()***	***toUpperCase()***	***valueOf()***

6.4 Objeto Number

JavaScript tem apenas um tipo de número**,** os quais podem ter ou não casas decimais.

JavaScript não é uma linguagem tipada, ao contrário de muitas outras linguagens de programação. O JavaScript não define diferentes tipos de números, como números int, short, long, float... Todos os números em JavaScript são armazenados como **float** de 64 bits(8 bytes).

JavaScript cria objetos do tipo **Number** quando uma variável é definido com um valor numerico, por exemplo var num = 255.336;. Raramente é necessário criar explicitamente objetos de **Number** .

Os números são convertidos em cadeias de caracteres em determinadas circunstâncias, por exemplo, quando um número é adicionado ou concatenado com uma String bem como utilizamos o método de **toString** .

O objeto do tipo **Number** tem suas próprias propriedades e métodos.

Atributos:

MAX_VALUE	**MIN_VALUE**	**NEGATIVE_INFINITIVE**	**POSITIVE_INFINITIVE**
NaN	**prototype**	**constructor**	

Métodos:

toExponential()	**toFixed()**	**toPrecision()**	**toString()**
valueOf()			

Atenção: consulte o nosso material de apoio para saber como utilizar cada método e atributo listado acima.

6.4.1 Atributos objeto Number - CONSTANTES

Objeto Number possui como atributos constantes: MAV_VALUE, MIN_VALUE, NEGATIVE_INFINITIVE, POSITIVE_INFINITIVE E NaN.

Para utilizar esses atributos, que são constantes, não precisamos criar ou instanciar um objeto do tipo Number como fizemos no Objeto String. Colocamos apenas a palavra-chave Number seguido de um ponto(.) e já conseguimos acessar os valores contidos nessa constante.

Veja um exemplo:

Number.MAX_VALUE	O maior número que pode ser representado em JavaScript. Retorna o valor: 1.7976931348623157e+308 .
Number.MIN_VALUE	o número mais próximo a zero que pode ser representado em JavaScript. Retorna o valor: 5.00E-324.

6.4.2 Representação de um número em Hexadecimal, Octal e notação científica

- **Hexadecimal:**

Podemos representar um número em hexadecimal em JavaScript colocando o valor: **0x** na frente do número a ser representado.

Exemplo:

```
var valor1 = 0x142;
```

- **Octal:**

Podemos também representar um número em Octal em JavaScript colocando o valor: **0** na frente do número a ser representado.

Exemplo:

```
var valor = 0377;
```

- **Notação científica:**

Para números muito grandes ou muito pequenos podemos representá-los usando a notação científica.

Veja abaixo como o JavaScript trabalha com esse tipo de notação:

```
var y = 123e5;  // 12300000
var z = 123e-5  // 0.00123
```

6.5 Objeto Math

Em JavaScript, podemos fazer uso de um objeto próprio para cálculos matemáticos chamado **Math** que possui constantes, métodos para ***calcular potências, raízes, arredondamentos, funções trigonométricas***, maneiras de encontrar o menor e o maior valor, além de um gerador de números randômicos. O objeto Math possui algumas constantes importantes para cálculos mais complexos, bem como métodos para executar operações matemáticas mais facilmente.

6.5.1 Constantes

O objeto Math possui 8 constantes que são:

E: constante do número de Euler. (2,718281828459045);

LN2: constante com o resultado do logaritmo natural na base 2. (0,6931471805599453); **LN10:** constante com o resultado do logaritmo natural na base 10. (2,302585092994046); **LOG2E:** constante com o resultado do logaritmo na base 2 do número de Euler.

(1,4426950408889634);

LOG10E: constante com o resultado do logaritmo na base 10 do número de Euler.

(0,4342944819032518);

PI: constante do pi (M). (3,141592653589793);

SQRT1_2: constante com o resultado da raíz quadrada de meio. (√½ = 0,7071067811865476);

SQRT2: constante com o resultado da raíz quadrada de 2 (√2 = 1,4142135623730951);

No caso, todas essas constantes são valores aproximados, levando-se em conta que são dízimas periódicas.

Abaixo está um exemplo de como obter o valor de todas as constantes.

```
<script type="text/javascript">
    document.write (Math.E +"<br>");
    document.write (Math.LN2 +"<br>");
    document.write (Math.LN10 +"<br>");
    document.write (Math.LOG2E +"<br>");
    document.write (Math.LOG10E +"<br>");
    document.write (Math.PI +"<br>");
    document.write (Math.SQRT1_2 +"<br>");
    document.write (Math.SQRT2 +"<br>");
</script>
```

6.5.2 Objeto Math e seus métodos

- **Raízes e Potências**

Podemos utilizar o objeto Math para obter raízes quadradas e potências. O método **sqrt()** extrai a raíz quadrada do número passado como argumento.

Exemplo:

```
<script type="text/javascript">
var1 = Math.sqrt(4);
document.write(var1);
</script>
```

Resultado: 2 (raiz quadrada de 4)

O método **pow()** retorna o valor da potência indicada em seus parâmetros, sendo o primeiro parâmetro o número base e o segundo o expoente.

```
<script type="text/javascript">
var1 = Math.pow(10,3); //Mesmo que 10³
document.write(var1); //Resultado: 1000
</script>
```

Exemplo:

Vejamos o código e o resultado logo abaixo:

Código:

```
<script type="text/javascript">
var var1 = 4;
var var2 = 2;
document.write("A raiz quadrade de "+ var1+" é " + Math.sqrt(var1)+"<br>");
document.write(var1+ " elevado a "+ var2 + " é " + Math.pow(var1,var2)+"<br>");
</script>
```

Saída:

A raiz quadrada de 4 é
24 elevados a 2 é 16

6.5.3 Arredondamentos

Quando tratamos com números que possuem a parte decimal extensa (como é o caso das constantes), podemos fazer uso de métodos para arredondar os números.

O método **round()** arredonda um número para o inteiro mais próximo, tanto para baixo quanto para cima. Por exemplo, o número 3.3 arredondado será 3, mas o número 3.8 arredondado será 4.

O método **floor()** arredonda um número para o inteiro mais baixo. Também considerado como piso. O método **ceil()** arredonda um número para o inteiro mais alto. Também considerado como teto.

O método **abs()** remove apenas a parte fracionada. Ou seja, retorna o valor absoluto.

Exemplo:

```
<script type="text/javascript">
var var1 = 4.5;
var var2 = -3.2;
document.write("O inteiro mais proximo de "+ var1+" é " + Math.round(var1)+"<br>");
document.write("O inteiro mais baixo(piso) de "+ var1+" é "+ Math.floor(var1)+"<br>");
document.write("O inteiro mais alto de "+ var1+" é " + Math.ceil(var1)+"<br>");
document.write("O valor absoluto de "+ var2+" é " + Math.abs(var2)+"<br>");
</script>
```

6.5.4 Trigonometria

Usado para cálculos trigonométricos envolvendo principalmente ângulos. Com esses métodos fica fácil obter o resultado matemático dos ângulos sem a necessidade de vários cálculos ou tabelas prontas.

sin(): retorna o valor de seno; **cos()**: retorna o valor de cosseno; **tan()**: retorna o valor da tangente; **asin()**: retorna o valor do arco seno;

acos(): retorna o valor do arco cosseno;

atan(): retorna o valor do arco tangente;

Exemplo:

```
<script type="text/javascript">
var var1 = 90;
document.write("O seno de "      + var1 + " é " + Math.sin(var1)+"<br>");
document.write("O cosseno "      + var1 + " é " + Math.cos(var1)+"<br>");
document.write("A tangente "     + var1 + " é " + Math.tan(var1)+"<br>");
document.write("O arco seno "    + var1 + " é " + Math.asin(var1)+"<br>");
document.write("O arco cosseno " + var1 + " é " + Math.acos(var1)+"<br>");
document.write("O arco tangente "+ var1 + " é " + Math.atan(var1)+"<br>");
</script>
```

6.5.5 Maior e Menor

Existem dois métodos do objeto Math que servem como comparativos. **min(valor1, valor2)**: retorna o menor valor entre os parâmetros passados. **max(valor1, valor2)**: retorna o maior valor entre os parâmetros passados.

Exemplo:

```
<script type="text/javascript">
var var1 = 7;
var var2 = 5;
document.write
    ("O maior valor entre "+var1+" e "+var2 + " é " + Math.max(var1,var2)+"<br>");
document.write
    ("O menor valor entre "+var1+" e "+var2 + " é " + Math.min(var1,var2)+"<br>");
</script>
```

6.5.6 Número Randômico

O objeto Math também possui um método para gerar automaticamente números randômicos.

O método **random()** retorna um número entre 0 e 1, ou seja, pode ser 0, 1, 0.5, 0.2, 0.8, 0.4567412, e assim por diante.

Se, por exemplo, quisermos fazer o limite entre 0 e 10, basta que multipliquemos por 10 o valor retornado por **random()**. Dessa forma conseguiremos um número entre randômico maior.

O problema de se usar isso é que os números retornados sempre serão muito fracionados, portanto, o ideal é utilizar junto uma das funções de arredondamento vistas nos tópicos anteriores.

Veja o exemplo abaixo:

```
<script type="text/javascript">
// Valor randomico
var valor1 = Math.random()*10;
var valor2 = Math.random()*10;
// Arredondando os valores
valor1 = Math.round(valor1);
valor2 = Math.round(valor2);
document.write
    ("Os numeros sorteados foram: "+valor1+" e "+valor2 );
</script>
```

```
Os números sorteados foram 4 e 2.
```

O que resulta em:

6.5.7 Objeto Boolean

Os objetos boolean servem para representar os valores booleanos (true/false). Foi acrescentado na versão 1.1 de JavaScript (com Netscape Navigator 3). Uma de suas possíveis características é a de conseguir valores booleanos a partir de dados de qualquer outro tipo.

Dependendo do que receba o construtor da classe Boolean o valor do objeto booleano que se cria será verdadeiro ou falso, da seguinte maneira:

- Inicia-se **false:** quando você não passa nenhum valor ao construtor, ou se passa uma cadeia vazia, o número 0 ou a palavra false sem aspas.

- Inicia-se **true:** quando recebe qualquer valor entre aspas ou qualquer número distinto de 0. Pode-se compreender o funcionamento deste objeto facilmente se examinarmos alguns exemplos.

```
<script type="text/javascript">
//Iniciando com FALSE (falso)
    var b1 = new Boolean()
    document.write(b1 + "<br>")

    var b2 = new Boolean("")
    document.write(b2 + "<br>")

    var b3 = new Boolean(false)
    document.write(b3 + "<br>")

    var b4 = new Boolean(0)
    document.write(b4 + "<br>")

//Iniciando com TRUE (verdadeiro)
    var b5 = new Boolean("Brasil")
    document.write(b5 + "<br>")

    var b6 = new Boolean(3)
    document.write(b6 + "<br>")
</script>
```

6.5.8 Objeto Date

O objeto Date é usado para trabalhar com datas e tempo. Para trabalhar com datas necessitamos instanciar um objeto da classe Date e com ele já podemos realizar as operações que necessitamos.

6.5.9 Criando um objeto Date

Um objeto da classe Date pode ser criado de duas maneiras distintas. Por um lado, podemos criar o objeto com o dia e hora atuais e por outro podemos criá-lo com um dia e hora distintos aos atuais. Isto depende

dos parâmetros que passemos ao construir os objetos.

Para criar um objeto com o dia e hora atuais, colocamos os parênteses vazios ao chamar ao construtor da classe Date. Veja exemplo:

```
<script type="text/javascript">
    minhaDataAtual = new Date();
    document.write(minhaDataAtual);
    //Resultado: Wed Dec 16 2015 18:50:57 GMT-0300 (Hora oficial do Brasil)
</script>
```

6.5.10 Definindo uma data específica

Claro que o "var data = new Date()" nem sempre é o que a gente quer. Muitas vezes, queremos que o objeto criado represente uma data específica, seja ela lida de um campo da tela, seja ela calculada de alguma forma.

Para criar um objeto data com um dia e hora diferentes dos atuais temos que indicar entre parênteses o momento para iniciar o objeto. Existem várias maneiras de expressar um dia e hora válida, por isso podemos construir uma data nos guiando por vários esquemas. Estes são dois deles, suficientes para criar todo tipo de datas e horas.

Código:

```
<script type="text/javascript">
    var ano = 2015;
    var mes = 12;
    var dia = 16;
    var hora = 18;
    var minutos = 50;
    var segundos = 30;
    minhaData1 = new Date(ano,mes,dia,hora,minutos,segundos);
    minhaData2 = new Date(ano,mes,dia);
    document.write("Definindo data 1: "+minhaData1.toString()+"<br>");
    document.writeln("Definindo data 2: "+minhaData2.toString());
</script>
```

Saída:

```
Definindo data 1: Sat Jan 16 2016 18:50:30 GMT-0300 (Hora oficial do Brasil)
Definindo data 2: Sat Jan 16 2016 00:00:00 GMT-0300 (Hora oficial do Brasil)
```

Os valores que devem ser passados para os construtores acima são sempre numéricos.

Um detalhe, o mês começa por 0(zero), ou seja, janeiro é o mês 0. Se não indicamos a hora, o objeto data

se cria com hora 00:00:00.

Assim é possível criar um objeto Date com o valor de qualquer data. Lembre-se de que o valor do mês parece sempre diminuído de 1, porque janeiro=0, fevereiro=1, março=2, ..., dezembro=11!! Até o argumento "dia" é obrigatório, os outros são opcionais.

6.5.11 Métodos do Objeto Date

O objeto Date não possui atributos/propriedades, por outro lado, possui muitos métodos. A seguir serão apresentados alguns dos seus principais métodos.

1. **getDate()**: devolve o dia do mês, um inteiro entre 1 e 31.
2. **getDay()**: devolve o dia da semana, inteiro entre 0 e 6 (0 para Domingo).
3. **getHours()**: retorna a hora, inteiro entre 0 e 23.
4. **getMinutes()**: devolve os minutos, inteiro entre 0 e 59.
5. **getSeconds()**: devolve os segundos, inteiro entre 0 e 59.
6. **getMonth()**: devolve o mês, um inteiro entre 0 e 11 (0 para Janeiro).
7. **getTime()**: devolve os segundos transcorridos entre o dia 1 de janeiro de 1970 e a data correspondente ao objeto ao que se passa a mensagem.
8. **getYear()**: retorna o ano, os últimos dois números do ano. Por exemplo, para o 2006 retorna 06. Este método está obsoleto em Netscape a partir da versão 1.3 de JavaScript e agora se utiliza getFullYear().
9. **getFullYear()**: retorna o ano com todos os dígitos com datas posteriores desde 2000.
10. **setDate(d)**: atualiza o dia do mês.
11. **setHours(h)**: atualiza a hora.
12. **setMinutes(m)**: muda os minutos.
13. **setMonth(m)**: muda o mês (atenção ao mês que começa por 0).
14. **setSeconds(s)**: muda os segundos.
15. **setTime(t)**: atualiza a data completa. Recebe um número de segundos desde 1 de janeiro de 1970.
16. **setYear(y)**: muda o ano, recebe um número, ao que lhe soma 1900 antes de colocá-lo como ano data. Por exemplo, se receber 95 colocará o ano 1995. Este método está obsoleto a partir de JavaScript 1.3

em

Netscape. Agora se utiliza setFullYear(), indicando o ano com todos os dígitos.

17. setFullYear(): muda o ano da data ao número que recebe por parâmetro. O número se indica completo ex: 2005 ou 1995. Utilizar este método para estar certo de que tudo funciona para datas posteriores a 2000.

18. getimezoneOffset(): Devolva a diferença entre a hora local e a hora GMT (Greenwich, UK Mean Time) sob a forma de um inteiro representando o número de minutos (e não em horas).

19. toGMTString(): devolva o valor do atual em hora GMT (Greenwich Mean Time)

Exercícios Propostos:

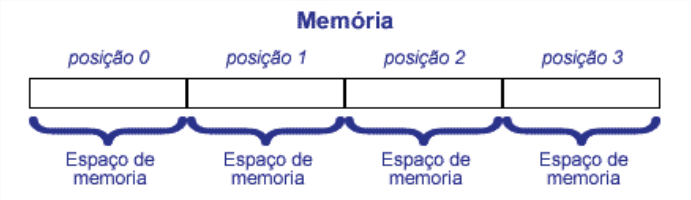
Leia a data de hoje e monte uma página com o calendário, destacando o dia de hoje (em vermelho, por exemplo).

Escreva um programa que receba uma data através de um campo de textos (prompt()) no formato dd/mm/aa. O programa deve reclamar (alert()) se o formato digitado for incorreto e dar uma nova chance ao usuário. Recebido o string, ele deve ser interpretado pelo programa que deverá imprimir na página quantos dias, meses e anos faltam para a data digitada.

7.0 Array

Nesse capítulo iremos abordar de forma clara as principais características de um array; Mostrar a sua criação e manipulações possíveis; Definir arrays multidimensionais ou matrizes; Determinar formas de interações e acessos.

Um array no JavaScript é atualmente um conjunto de valores ordenado por índices. Podemos relacionar cada valor com um índice/chave, para indicar em qual posição um determinado valor está armazenado dentro do array.



Ele é otimizado de várias maneiras, então podemos usá-lo como um array real, lista (vetor), hashtable (que é uma implementação de mapa), dicionário, coleção, pilha, fila e provavelmente muito mais. Além disso, JavaScript nos oferece uma gama enorme de funções para manipulá-los.

A explicação dessas estruturas está além do escopo dessa apostila, mas todo conteúdo aqui abordado traz uma boa base para quem estar iniciando o conteúdo de array.

7.1 Criando um Array

Existe várias formas de construção de um array em JavaScript, veja:

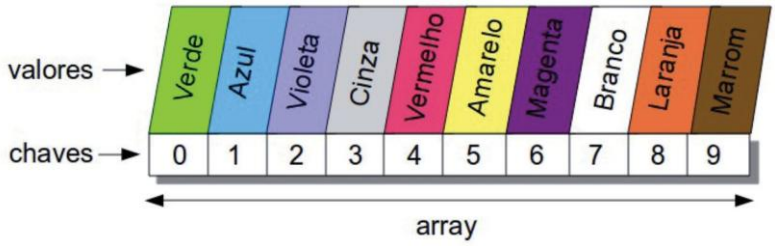
- Construção simples sem dimensionamento: Quando criamos um array, mas não especificamos quantas posições ele irá possuir;
- **Construção simples com dimensionamento**: Quando criamos um array e especificamos quantas posições ele irá possuir. Estas posições são vazias.
- Construção inserindo valores: Quando criamos um array e especificamos exatamente quais valores ele irá possuir. Cada valor deve ser separado por vírgula e qualquer tipo de dado pode ser utilizado para popular nosso vetor.

Veja um exemplo de cada forma de criar um vetor/array:

```
<script type="text/javascript">
    //Sem dimensionamento
    meuVetor1 = new Array();

    //Com dimensionamento
    meuVetor2 = new Array(4);

    //Com valores inseridos
    meuVetor3 = new Array("Raquel","Fabiana","Diana","Alana");
</script>
```

Detalhe importante é que todo array não associativo começa pela chave ou índice de número 0.

A figura abaixo representa um array que tem como valor representação de cores, e possui dez posições,



cada posição representa uma cor, seu índice (chave) vai de 0 até 9. Veja:

Em código temos:

```
<script type="text/javascript">
    var cores = new Array("verde","azul","violeta","cinza",
        "vermelho","amarelo","magenta","branco","laranja","marrom");
</script>
```

7.2 Arrays associativos.

Os arrays associativos associa-se um determinado valor ou nome a um dos valores do array.

O array associativo usa strings como índice, onde cada string pode representas uma chave.

Observe a sintaxe:

```
<script type="text/javascript">
    var vetor = {"chave1":"valor1","chave2":"valor2","chaveN":"valorN",}
</script>
```

Observe que quando usamos arrays associativos, a compreensão é mais fácil, dando mais legibilidade ao código. Porém, não é adequado utilizá-lo dentro de um loop tradicional(while,do..while,for), por isso temos um loop especial para percorrer arrays associativos chamado de for..in.

```
<script type="text/javascript">
//Forma 01
    var vetor1 = {"nome":"Valdenia","rua":"São João","bairro":"Messejana"};
//Forma 02
    var vetor2 = new Array[3];
    vetor2["nome"] = "Valdenia";
    vetor2["rua"] = "São João";
    vetor2["bairro"] = "Messejana";
</script>
```

Veja um exemplo de array associativo escrito de duas formas diferentes:

Umas das vantagens do array associativo é quando fazemos o acesso ao array, onde temos de forma clara e compreensível o valor que aquela chave pode conter. Como por exemplo nome, onde só vai existir o nome de pessoas. Veja abaixo um exemplo de acesso aos valores armazenados em um array dessa natureza.

Exemplo:

```
<script type="text/javascript">
    var vetor = new Array(3);
    vetor['nome'] = "Valdenia";
    vetor['rua'] = "São João";
    vetor['bairro'] = "Messejana";

    document.writeln(vetor['nome']);
    document.writeln(vetor['rua']);
    document.writeln(vetor['bairro']);
</script>
```

Dessa forma podemos acessar o array. Basta determinar o nome do array e qual a chave, onde cada chave tem um valor já determinado. Resultará em um erro o uso de uma chave errada.

7.3 Interações

Quando falamos de interações em um array estamos dizendo o mesmo que percorrer esse array usando mecanismos da própria linguagem. Como isso as interações podem ser feitas de várias formas, mas no JavaScript podem ser iterados pelo operador FOR(chave IN array) que já vimos anteriormente.

Exemplo:

```
<script type="text/javascript">
    var vetor = new Array(3);
    vetor['nome'] = "Valdenia";
    vetor['rua'] = "São João";
    vetor['bairro'] = "Messejana";

    for (chave in vetor){
        document.write(chave + " = " + vetor[chave] + "<br>");
    }
</script>
```

Resultado:

nome = Valdenia
rua = São João
bairro = Messejana

Esse tipo de interação é muito utilizado, principalmente quando temos arrays associativos.

7.4 Acessando um Array

Quando criamos um array temos que ter em mente que estamos criando uma variável que possui vários valores e que os mesmo podem ser acessados a qualquer momento. Cada valor está guardado em uma posição que pode ser acessada através de uma chave.

A sintaxe para acesso simplificado de um array é a seguinte:

```
nome_do_array[chave_de_acesso] ;
```

Temos que ter cuidado ao passar uma chave para o array, pois ela deve conter o mesmo nome de qualquer umas das chaves existentes no array. Caso a chave não exista, o valor não poderá ser resgatado. A sintaxe acima retorna um valor contido no array, por esse motivo temos que atribuir esse valor como mostra o exemplo abaixo:

```
<script type="text/javascript">
//Array com Valores
    var meu_array = new Array("nome","telefone","rua");
// Acessando uma posição
    var elemento_array = meu_array[1];

document.write(elemento_array);
</script>
```

Resultado: telefone.

7.5 Alterando um Array

Podemos alterar qualquer valor de um array. É muito semelhante ao acesso, onde, a diferença está na chamada do array. É nesse momento que atribuímos um novo valor.

nome_do_array[chave_de_acesso] = <novo_valor> ;

Observe o exemplo abaixo:

```
<script type="text/javascript">
    //criando um array com valores
    var meu_array = new Array("nome","telefone","rua","cidade");

    //alterando o valor de uma posições
    meu_array[1] = "sobrenome";

    //imprime na tela.
    for(chave in meu_array){
        document.write(meu_array[chave]+"<br>");
    }
</script>
```

Resultados:

nome
sobrenome
rua
cidade

Vimos no exemplo anterior o valor da posição 1 do array ('telefone') foi alterada para sobrenome. Vale ressaltar que esse array tem suas chaves definidas de forma automática. A primeira posição é 0, a segunda é 1, e assim sucessivamente. Veja mais um exemplo onde alteramos o valor, mas usando o operador "+=":

```
<script type="text/javascript">
    //criando um array vazio
    var produto = new Array();
    //adicionando valores
    produto["nome_produto"] = "Arroz";
    produto["valor_produto"] = 1.35;

    //alterando valor do pruduto
    produto["valor_produto"] += .63;

    //alterando nome do produto
    produto["nome_produto"] = " Tio João ";

    //imprime na tela.
    for(chave in produto){
        document.write(produto[chave]+"<br>");
    }
</script>
```

Podemos observar que assim como as variáveis “comuns”, a forma de alterar o valor de um array é igual. A diferença está na chamada do elemento de um array, pois temos que passar a chave além do valor que queremos atribuir.

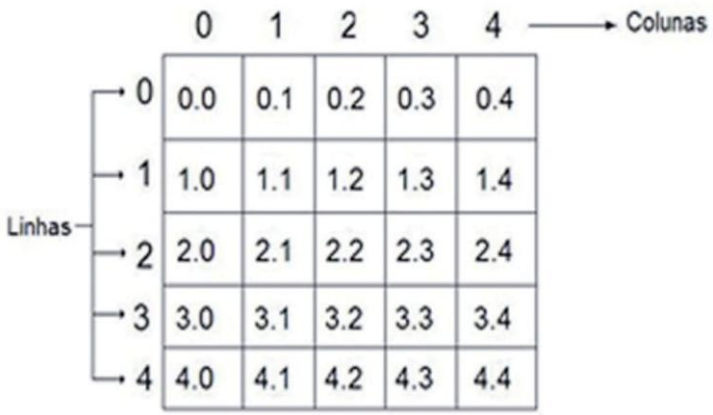
7.6 Arrays multidimensionais

Os arrays multidimensionais são estruturas de dados que armazenam os valores em mais de uma dimensão. Os arrays que vimos até agora armazenam valores em uma dimensão, por isso para acessar às posições utilizamos somente um índice ou chave. Os arrays de 2 dimensões salvam seus valores de alguma forma como em filas e colunas e por isso, necessitaremos de dois índices para acessar a cada uma de suas posições.

Em outras palavras, um array multidimensional é como um “contêiner” que guardará mais valores para cada posição, ou seja, como se os elementos do array fossem por sua vez outros arrays.

Outra ideia que temos é que matrizes são arrays nos quais algumas de suas posições podem conter outros arrays de forma recursiva. Um array multidimensionais pode ser criado pela função array():

Na figura abaixo temos a representação de um array com duas dimensões.



Uma diferença importante de um array comum para um multidimensional é a quantidades de chaves (índices), onde cada um dos índices representa uma dimensão.

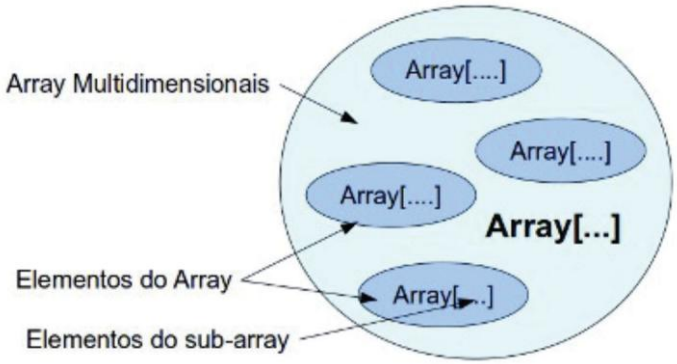
Código:

```
<script type="text/javascript">
    var matriz = new Array(
                new Array("0.0 " , "0.1 " , "0.2 " , "0.3 ") ,
                new Array("1.0 " , "1.1 " , "1.2 " , "1.3 ") ,
                new Array("2.0 " , "2.1 " , "2.2 " , "2.3 ") ,
                new Array("3.0 " , "3.1 " , "3.2 " , "3.3 ") );
</script>
```

Outra forma de iniciar o array:

```
<script type="text/javascript">
    var matriz = new Array(4,4);
    matriz[0] = new Array("0.0 " , "0.1 " , "0.2 " , "0.3 ");
    matriz[1] = new Array("1.0 " , "1.1 " , "1.2 " , "1.3 ");
    matriz[2] = new Array("2.0 " , "2.1 " , "2.2 " , "2.3 ");
    matriz[3] = new Array("3.0 " , "3.1 " , "3.2 " , "3.3 ");
</script>
```

Observe que temos uma chave para representar a linha e outra para representar a coluna, assim, determinando uma matriz 4x4. Podemos ver também que inicializamos um array dentro do outro. Cada sub- array é uma linha, e cada elemento do array maior representa as colunas.



Para acessarmos o valor de um array multidimensional, basta colocar as duas ou mais chaves da posição que queremos acessar. É muito semelhante ao array de uma única dimensão.

Observe o acesso aos exemplos anteriores:

```
nme_do_array[chave_1][chave_2]...[chave_n];
```

Exemplo :

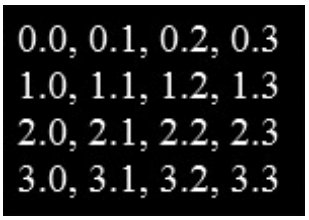
```
<script type="text/javascript">
    var matriz = new Array(4,4);
    matriz[0] = new Array("0.0 " , "0.1 " , "0.2 " , "0.3 ");
    matriz[1] = new Array("1.0 " , "1.1 " , "1.2 " , "1.3 ");
    matriz[2] = new Array("2.0 " , "2.1 " , "2.2 " , "2.3 ");
    matriz[3] = new Array("3.0 " , "3.1 " , "3.2 " , "3.3 ");

    //acessando array linha 1 coluna 2
    document.write(matriz[1][2]); //Resultado: 1.2
</script>
```

Dessa forma podemos acessar o elemento “1.2 “ que está guardado na posição linha 1 coluna 2, lembrando que o primeiro elemento de um array é 0. Abaixo, um exemplo que acessa todos os valores do array e imprime com quebra de linha:

Código:

```
<script type="text/javascript">
    var matriz = new Array(
                new Array("0.0" , "0.1" , "0.2", "0.3") ,
                new Array("1.0" , "1.1" , "1.2" , "1.3") ,
                new Array("2.0" , "2.1" , "2.2" , "2.3") ,
                new Array("3.0" , "3.1" , "3.2" , "3.3") );
    //percorrendo matriz
    for(i=0 ; i<matriz.length;i++){
        for(j=0 ; j<matriz[i].length;j++){
            //retirando a ultima virgula da linha na hora da impressão
            if(j != matriz[i].length-1){
                document.write(matriz[i][j]+", ");
            }
            else{
                document.write(matriz[i][j]);
            }
        }
        document.write("<br>");
    }
</script>
```

Resultado:



Explicando o código:

Linha 325 à 329 – criamos um array de duas dimensões.

Linha 331 – temos um for que irá percorrer as linhas de nossa matriz. Esse for tem uma variável i iniciando com 0 e será incrementado de um em um enquanto i for menor que o número de linhas da matriz

Linha 332 – temos um for que irá percorrer as colunas de nossa matriz. Esse for tem uma variável j iniciando com 0 e será incrementado de um em um enquanto j for menor que o número de colunas da matriz

Linha 334 –temos um if que irá verificar se não estamos na última coluna da linha que está sendo impressa. Se não estiver na última coluna irá imprimir o valor da coluna concatenado com uma “, ”.

Linha 337 – caso a condição da linha 334 não seja verdade, é porque iremos imprimir a última coluna da linha que está sendo impressa, então irá imprimir somente o valor da coluna, sem a vírgula.

Linha 341 – temos uma quebra de linha, que será executa todas as vezes que terminarmos de imprimir todos os valores de uma linha, ou seja quando o for mais interno for finalizado

7.6 Funções com Arrays

O objeto Array possui uma propriedade chamada length. Esta propriedade mostra quantos elementos o vetor possui.

Código:

```
<script type="text/javascript">
    vetor = new Array("João", "Roberto","José");
    document.write(
        "<p>Este vetor possui "+vetor.length+" elementos. Que são: </p>");
    document.write("<ul>");
    for (indice = 0; indice < vetor.length; indice++) {
        document.write ("<li>"+ vetor [indice]+ "</li>")
    };
    document.write("</ul>");
</script>
```

Saída:

Este vetor possui 3 elementos. Que são: João
Roberto José

7.7Junção e Concatenação de Arrays (Vetores)

Como vimos anteriormente, os vetores em JavaScript são constituídos pelo objeto Array.

Há momentos que queremos unir um ou mais vetores. JavaScript nos possibilita fazer isso resultando duas formas diferentes.

7.8 Método join()

O vetor construído com o objeto Array de JavaScript pode ter a junção de seus elementos. A junção consiste em criar uma string única usando separadores.

O método **join()**nos possibilita a junção de forma simples, precisa e correta em uma string.

Por exemplo, vamos imaginar que temos um vetor que possui os dados de uma data qualquer em seus índices e que queremos mostrar a saída dessa data inteira. Há duas formas que podemos fazer isso e são complicadas.

Uma delas é criar um loop entre os dois primeiros índices e depois mostrar apenas o último índice.

```
<script type="text/javascript">
    var data = new Array(3);
    data[0] = 27;
    data[1] = 3;
    data[2] = 2009;

    for (i = 0; i<data.length-1; i++) {
        document.write (data[i] + "/");
    }
    document.write (data[2]);
</script>
```

A outra forma, apesar de mais simples, ainda não é a mais indicada.

```
<script type="text/javascript">
    var data = new Array(3);
    data[0] = 27;
    data[1] = 3;
    data[2] = 2009;
    document.write (data.join("/"));
</script>
```

Vale lembrar que o método join não modifica o vetor original, mas é possível guardar seu resultado em uma variável. Ex.: dataCompleta = data.join("/").

7.8.1 Método concat()

O método **concat()**consiste em unir um ou mais arrays (vetores). Este método usa como argumento um objeto do tipo array. Se desejarmos unir mais de um vetor, cada objeto array do argumento deve vir separado por vírgula (,).

Código:

```
<script type="text/javascript">
    var alunosAno1 = new Array ("André","Sansão","Raquel");
    var alunosAno2 = new Array ("Marcos","Valdenia","Fabiana");
    var alunosAno3 = new Array ("Thiago","Elinardy","Krystian");
    var alunosAno4 = new Array ("Fabricio","Patricio","Atila");
    var todosAlunos = alunosAno1.concat(alunosAno2,alunosAno3,alunosAno4);
    document.write ("Esta escola tem "+ todosAlunos.length +
        " alunos no projeto e-Jovem.<br>" + "Eles se chamam:<br>"+ todosAlunos);
</script>
```

Aqui, o resultado obtido na variável todosAlunos é o nome contido nas quatro variáveis alunosAno.

Saída:

Esta escola tem 12 alunos no projeto e-Jovem. Eles se chamam:

André,Sansão,Raquel,Marcos,Valdenia,Fabiana,Thiago,Elinardy,Krystian,Fabricio,Patricio,Atila

Assim como no método join(), concat() também não altera o conteúdo original dos vetores.

7.8.2 Método push()

Além da forma algorítmica de inserir um elemento no final de um array, o próprio objeto Array possui um método que faz isso por nós.

O método push() insere um ou mais elementos no final de um array. A sintaxe é: push (elemento1, elemento2, elemento3, ...)

Dessa forma, não precisamos nos preocupar com o tamanho e nem com o último índice do array.

Código:

```
<script type="text/javascript">
    vetor = new Array ("Cristiane");
    document.write ("Este array possui "+vetor.length+" elementos.<br>");
    vetor.push ("Nogueira");
    document.write("Agora possui "+vetor.length+" elementos.<br>");
    vetor.push ("Wallison","Ledruwick","Crispiano","Mateus");
    document.write ("Este vetor possui "+vetor.length+
        " elementos.<br>"+"Que são: "+vetor.join(", "));
</script>
```

Saída:

Este array possui 1 elementos.
Agora possui 2 elementos.
Este vetor possui 6 elementos.
Que são: Cristiane, Nogueira, Wallison, Ledruwick, Crispiano, Mateus

7.8.3 Método unshift()

Outra forma de inserir elementos é utilizar o método unshift(). O método unshift() insere um ou mais elementos no início de um array.

A sintaxe é: unshift (elemento1, elemento2, elemento3, ...)

Um detalhe importante é o fato de que este método possivelmente não funcione corretamente em versões

anteriores ao Internet Explorer 6.

Código:

```
<script type="text/javascript">
    vetor = new Array ("5","6","7","8","9","10","J","Q","K");
    document.write ("Este baralho tem apenas as cartas "+vetor.join(", ")+
        ". Vamos completá-lo com as cartas iniciais. <br>");
    vetor.unshift ("A","1","2","3","4");
    document.write("Agora temos o baralho completo: "+vetor.join(", "));
</script>
```

Saída:

Este baralho tem apenas as cartas 5, 6, 7, 8, 9, 10, J, Q, K. Vamos completá-lo com as cartas iniciais.
Agora temos o baralho completo: A, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K

7.8.4 Método pop()

O método **pop()** remove um elemento do final de um array e retorna o conteúdo do índice removido.

Não precisamos nos preocupar com o tamanho e nem com o último índice do array, pois pop() fará todo o trabalho sozinho.

Código:

```
<script type="text/javascript">
    vetor = new Array ("Marcelo","Patricia","Joelma","Diana");
    document.write ("Este array possui "+vetor.length+"elementos.<br>");
    document.write ("Vamos remover "+vetor.pop()+ " da lista.<br>");
    document.write ("Agora, temos "+vetor.length+" elementos. <br>"+
        "Que são: "+vetor.join(", "));
</script>
```

Saída:

Este array possui 4 elementos.
Vamos remover Diana da lista.
Agora, temos 3 elementos.
Que são: Marcelo, Patricia, Joelma

7.8.5 Método shift()

Outra forma de remover elementos é utilizar o método shift(). O método shift()remove um elemento do início de um array e retorna o conteúdo do índice removido.

Código:

```
<script type="text/javascript">
    vetor = new Array ("Marcelo","Patricia","Joelma","Diana");
    document.write ("Este array possui "+vetor.length+" elementos.<br>");
    document.write ("Vamos remover "+vetor.shift()+ " da lista.<br>");
    document.write ("Agora, temos "+vetor.length+" elementos. <br>"+
        "Que são: "+vetor.join(", "));
</script>
```

Saída:

Este array possui 4 elementos.
Vamos remover Marcelo da lista.
Agora, temos 3 elementos.
Que são: Patricia, Joelma, Diana

7.8.6 Método reverse()

O método `reverse` do objeto array serve para inverter a ordem dos elementos de um array. Dessa forma podemos usar apenas uma forma de organização e depois inverter o array para conseguir a ordem desejada.

Código:

```
<script type="text/javascript">
    var vetor = Array (1,2,3,4,5);
    document.write ("Array inicial:" + vetor);
    document.write ("<br> Array invertido: "+ vetor.reverse());
</script>
```

Saída:

Array inicial:1,2,3,4,5
Array invertido: 5,4,3,2,1

7.8.6 Método sort()

O método çort é o que realmente faz a organização do array. Porém, **sort()** apenas organiza o array de forma alfabética, se quisermos organizar o array de forma numérica devemos criar uma função para isso.

Abaixo está um exemplo de uma organização simples feita em ordem alfabética.

Código:

```
<script type="text/javascript">
    vetor = new Array ("Elinny","André","Adriano","Elisabete");
    document.write (vetor.sort());
</script>
```

Veja que *vetor* é organizado corretamente de forma alfabética.

Saída:

Adriano,André,Elinny,Elisabete

Agora, podemos perceber que o resultado da organização de números não é dado de forma satisfatória.

Código:

```
<script type="text/javascript">
    vetor = new Array (3000,20,100,4);
    document.write (vetor.sort());
</script>
```

Saída:

```
100,20,3000,4
```

Exercícios Resolvido

1. Quais serão as letras que serão geradas a partir desse código?

```
<script type="text/javascript">
    var valor = 2;
    var indice  = valor;
    var letras = new Array('d','b','e','a','i','o','r');

    document.write(letras[indice++]+" ");
    document.write(letras[indice]+" ");
    document.write(letras[indice-2]+" ");
    document.write(letras[0]+" ");
    document.write(letras[valor+1]+" ");
</script>
```

Entendendo o algoritmo

- No escopo temos uma variável chamada “valor” = 2 e “índice” que recebe o conteúdo que está em valor que é 2 também.

- Temos também um array que irá receber um conjunto descrito, lembrando que o índice inicial de um vetor é zero até a sua dimensão menos um. Nesse caso temos 7 elementos, então os índices para o meu vetor são de zero até 6.

- Para cada impressão, temos uma letra seguido de uma concatenação com um espaço em branco, então podemos chegar a conclusão que irá ser impresso um conjunto de letras com e espaços.
- O primeiro “document.write” irá imprimir o valor de “índice” = 2 que é representado pela letra “e”, em seguida índice será incrementado passando a ser igual a 3.
- O segundo “document.write” irá imprimir o valor de índice = 3 que é representado pela letra “a”.
- O terceiro “document.write” irá imprimir o valor de (índice - 2) = (3-2) = 1 que é representado pela letra “b”
- O quarto “document.write” irá imprimir o índice zero do meu array representado pela letra “d”
- O quinto “document.write” irá imprimir o índice (valor +1) = (2 + 1) = 3 representando pela letra “a”
- Por fim podemos identificar a mensagem no JAVASCRIPT: e a b d a.

Exercícios Propostos:

O que é um array, e qual a sua principal finalidade? Declare um array chamado “nomes” com 8 posições, e grave nomes de pessoas que você conhece em cada uma delas. Após criar o array responsa as questões 2, 4, 5, 10, 11:

Utilizado o array responda.

a) Qual nome é impresso no navegador se colocarmos o código:

```
document.write(nomes[3]);
```

b) Quais nomes aparecerá se adicionamos os seguintes códigos:

```
for(i= 6; i>1 ; i--){ document.write(nomes[i]); }
```

c) O que acontece se chamarmos uma posição que não existe, exemplo: nomes[15];

O que é um array associativo, de exemplos:

Usando o comando *for...in*, crie uma interação onde todos os nomes possa ser impresso na

tela.

Utilizando o mesmo código, altere alguns nomes do array.

O que é um array multidimensional?

Crie um array “palavras” multidimensional 5x3 com os valores da tabela abaixo, e responda as questões 7,8,9:

“oi”	“tudo”	“estar”
“você”	“vai”	“?”
“com”	“dia”	“!”
“,”	“bem”	“sim”
“casa”	“hoje”	“em”

Crie um código JAVASCRIPT onde com os valores do array possa ser impresso na tela com a frase “oi, tudo bem com você?”.

Utilizando as posições da sequência [1][0],[1][1],[0][2],[4][2],[4][0],[4][1],[1][2] do array palavras, qual frase podemos formular? Utilize a função document.write() para mostrar na tela do navegador.

Construa um código JAVASCRIPT para mostra uma resposta para a pergunta da questão 7. (Use o comando document.write para imprimir na tela).

Utilizando a função sort(), imprima em ordem alfabética os nomes do array “nomes”.

Use o comando unshift() para adicionar mais dois nomes no array.

Crie um documento HTML com um código JavaScript que armazene 10 números inteiros em um array. Preencha todas as posições do array com valores sequenciais e em seguida imprima-os na tela. Em seguida, escolha duas posições aleatoriamente e troque os valores de uma posição pelo da outra. Repita essa operação 10 vezes. Ao final, imprima o array novamente.

Crie um documento HTML com um código JavaScript que armazene 10 números inteiros em um array. Preencha todas as posições com valores aleatórios e em seguida imprima-os. Ordene do menor valor para o maior e imprima novamente.

8.0 Manipulação de Funções

Quando queremos um código funcional para determinado fim, com por exemplo fazer um cálculo ou

alguma interação dentro do JavaScript, usamos o que chamamos de função. As funções são um pedaço de código com o objetivo específico, encapsulado sob uma estrutura única que recebe um conjunto de parâmetros e retorna ou não um determinado dado. Uma função é declarada uma única vez, mas pode ser utilizada diversas vezes. É uma das estruturas mais básicas para prover reusabilidade ou reaproveitamento de código, deixando as funcionalidades mais legíveis.

8.1 Declarando uma Função.

Declaramos uma função, com o uso do operador function seguido do nome que devemos obrigatoriamente atribuir, sem espaços em branco e iniciando sempre com uma letra. Temos na mesma linha de código a declaração ou não dos argumentos pelo par de parênteses "()". Caso exista mais de um parâmetro, usamos virgula(,) para fazer as separações. Logo após encapsulamos o código pertencente a função por meio das chaves ({}). No final, temos o retorno com o uso da cláusula **return** para retornar o resultado da função que pode ser um tipo inteiro, array, string, ponto flutuante etc. A declaração de um retorno não é obrigatório. Observe a sintaxe:

```
function nome_da_função( argumento_1, argumento_2, argumento_n )
{
comandos; return $valor;
}
```

Observe um exemplo onde criamos uma função para calcular o índice de massa corporal de uma pessoa (IMC), onde recebe como parâmetro dois argumentos. Um é a altura representada pela variável

$altura e o outro é o peso representada pela variável $peso. Passamos como parâmetros para essa função o peso = 62 e a altura = 1.75. Observe:

```
<script type="text/javascript">
    //Declarando a função
    function calculo_IMC(peso, altura){
        return peso/(altura*altura);
    }
    //Chamando a função
    document.write(calculo_IMC(62,1.75));
</script>
```

Exemplo:

Resultado: 20.244897959183675

Nesse exemplo temos a declaração e logo após a chamada da função, onde é nesse momento que passamos os dois parâmetros na ordem que foi declarada na função. Lembrando que essa ordem é obrigatória. Observe mais um exemplo, onde a função declarada, porém não possui a cláusula **return.**

```
<script type="text/javascript">
    //Declarando a função
    function imprime(texto){
        document.write("<h1>"+texto+"</h1>");
    }
    //Chamando a função
    imprime("Testando a função.");
</script>
```

8.2 Escopo de Variáveis em Funções

Um conceito importante em programação são os tipos de declarações de variáveis, onde sua visibilidade vai depender de onde ela é declarada. O acesso a essas variáveis pode ser definido da seguinte forma:

Variáveis locais < São aquelas declaradas dentro de uma função e não tem visibilidade fora dela.

Variáveis Globais < São variáveis declaradas fora do escopo de uma função, porém tem visibilidade (pode ser acessada) ao contexto de uma função sem passá-la como parâmetro. Para isso declaramos a variável no escopo fora da função e fazemos a sua chamada dentro da função.

8.3 Valor de Retorno

Toda função pode opcionalmente retornar um valor, ou simplesmente executar os comandos e não retornar valor algum. Não é possível que uma função retorne mais de um valor, mas é permitido fazer com que uma função retorne um valor composto, como listas ou array's. As operações aritméticas podem ser feitas de forma direta no retorno. Observe um exemplo onde temos uma operação direta:

```
<script type="text/javascript">
    function somar(n1,n2){
        return n1+n2;
    }
    function texto () {
        return "O resultado é: ";
    }
    document.write(texto()+ somar(115,23));
    //Resultado: 138
</script>
```

Também podemos determinar mais de um retorno desde que eles não sejam acessados ao mesmo tempo, observe o exemplo:

```
<script type="text/javascript">
    function teste(caso){
        switch(caso){
            case 1:
                return "opção 1";
            case 2:
                return "opção 2";
            case 3:
                return "opção 3";
            default:
                return null;
        }
    }
    alert(teste(2));
</script>
```

Esse código mostra de forma clara que não existe a possibilidade de retornarmos mais de um **return**, caso isso ocorresse, teríamos um erro, ou não funcionamento da função.

8.4 Recursão.

Função recursiva é uma definição usada tanto na programação quanto na matemática, onde, significa que uma função "faz a chamada" de si mesma na sua execução. Um exemplo é o cálculo do fatorial de um número. Observe:

```
<script type="text/javascript">
    function fatorial(numero){
        if (numero === 1) {
            return numero;
        }else{
            return numero * fatorial(numero-1);
        }
    }
    alert(fatorial(5)); //Resultado: 120
</script>
```

Fatorial de 5: *5! = 5*4!, 4! = 4*3!, 3! = 3*2!, 2! = 2*1! ou 5*4*3*2*1 = 120.*

Exercícios Propostos

Diga com suas palavras uma definição para função, e como podemos declará-la em PHP.

Qual a diferença de variáveis globais para variáveis locais e como podemos defini-las em PHP?

Quais as funções que podemos usar para criarmos uma função onde seus parâmetros são passados pro argumentos variáveis?

O que é um valor de retorno e qual o comando usado quando queremos retornar algo dentro de uma função?

O que é recursão?

Crie uma função que determine se um numero é par ou ímpar. E faça uma chamada dessa função imprimindo o resultado.

Crie uma função que calcule a fatorial de um número.

Crie uma função para determina se um numero é primo ou não. Numero primo é aquele que possui dois divisores, 1 e ele mesmo. Criem um laço de repetição e use estrutura de controle.

9.0 Eventos JavaScript

Quando um usuário visita uma página web e interage com ela se produzem os eventos e com

JavaScript podemos definir o que queremos que ocorra quando se produzam.

Com JavaScript podemos definir o que acontece quando se produz um evento, como por exemplo quando um usuário clica sobre um botão, edita um campo de texto ou abandona a página um evento é lançado e podemos definir que alguma função seja executa quando isso acontecer.

9.1 Relação de eventos

Abaixo uma lista de eventos suportados pelo JavaScript com suas respectivas descrições e a partir de que versão do JavaScript esse evento foi incorporado.

- **onabort**

Este evento se produz quando um usuário cancela o carregamento de uma imagem, seja porque para o carregamento da página ou porque realiza uma ação que a detém, como por exemplo, sair da página. JavaScript 1.1

- **onblur**

Desata-se um evento onblur quando um elemento perde o foco da aplicação. O foco da aplicação é o lugar onde está situado o cursor, por exemplo, pode estar situado sobre um campo de texto, uma página, um botão ou qualquer outro elemento. JavaScript 1.0

- **onchange**

Desata-se este evento quando muda o estado de um elemento de formulário, às vezes não se produz até que o usuário retire o foco da aplicação do elemento. JavaScript 1.0

- **onclick**

Produz-se quando se clica o botão do mouse sobre um elemento da página, geralmente um botão ou um link. JavaScript 1.0

- **ondragdrop**

Produz-se quando um usuário solta algo que havia arrastado sobre a página web. JavaScript 1.2

- **onerror**

Produz-se quando não se pode carregar um documento ou uma imagem e esta fica quebrada. JavaScript 1.1

- **onfocus**

O evento onfocus é o contrário de onblur. Produz-se quando um elemento da página ou a janela ganham o

foco da aplicação. JavaScript 1.0

- **onkeydown**

Este evento é produzido no instante que um usuário pressiona uma tecla, independentemente que a solte ou não. É produzido no momento do clique. JavaScript 1.2

- **onkeypress**

Ocorre um evento onkeypress quando o usuário deixa uma tecla clicada por um tempo determinado. Antes deste evento se produz um onkeydown no momento que se clica a tecla. JavaScript 1.2

- **onkeyup**

Produz-se quando o usuário deixa de apertar uma tecla. É produzido no momento que se libera a tecla. JavaScript 1.2

- **onload**

Este evento se desata quando a página, ou em JavaScript 1.1 as imagens, terminaram de se carregar. JavaScript 1.0

- **onmousedown**

Produz-se o evento onmousedown quando o usuário clica sobre um elemento da página. onmousedown se produz no momento de clicar o botão, soltando ou não.JavaScript 1.2

- **onmousemove**

Produz-se quando o mouse se move pela página. JavaScript 1.2

- **onmouseout**

Desata-se um evento onmuoseout quando a seta do mouse sai da área ocupada por um elemento da página. JavaScript 1.1

- **onmouseover**

Este evento desata-se quando a seta do mouse entra na área ocupada por um elemento da página. JavaScript 1.0

- **onmouseup**

Este evento se produz no momento que o usuário solta o botão do mouse, que previamente havia clicado. JavaScript 1.2

- **onmove**

Evento que se executa quando se move a janela do navegador, ou um frame. JavaScript 1.2

- **onresize**

Evento que se produz quando se redimensiona a janela do navegador, ou o frame, no caso de que a página

os tenha. JavaScript 1.2

- **onreset**

Este evento está associado aos formulários e se desata no momento que um usuário clica no botão de reset de um formulário. JavaScript 1.1

- **onselect**

Executa-se quando um usuário realiza uma seleção de um elemento de um formulário. JavaScript 1.0

- **onsubmit**

Ocorre quando o visitante aperta sobre o botão de enviar o formulário. Executa-se antes do envio propriamente dito. JavaScript 1.0

- **onunload**

Ao abandonar uma página, seja porque se clique em um link que nos leva a outra página ou porque se fecha a janela do navegador, se executa o evento onunload.

Destes eventos iremos selecionar e estudar os principais nos próximos tópicos

9.2 Evento onclick

O evento onclick é lançado/disparado quando clicamos em um elemento da página, geralmente um botão ou um link. Esse evento é valido para os objetos ou componentes de página HTML: Button, Checkbox, Radio, Link, Reset e Submit.

Sintaxe

```
<elemento onclick="AlgumCodigoJavaScript">
```

Em HTML:

Exemplo: onclick

Quando o usuário clicar no botão “Clique aqui” o evento onclick será disparado e chamara automaticamente a função de nome “minhaFuncao” a qual irá imprimir a seguinte String em um caixa de diálogo alert: “O evento onclick foi disparado”.

```html
<!DOCTYPE html>
<html lang="pt">
    <head>
        <meta charset="utf-8">
        <title>
            Onclick
        </title>

        <script type="text/javascript">
            function minhaFuncao(numero){
                alert("Evento Onclick");
            }
        </script>
    </head>

    <body>
        <p>Clique no botão </p>
        <button onclick="minhaFuncao()">Clique Aqui</button>
    </body>
</html>
```

O mesmo exemplo poderia ser implementado dessa forma:

```html
<!DOCTYPE html>
<html lang="pt">
    <head>
        <meta charset="utf-8">
        <title>
            Onclick
        </title>
    </head>

    <body>
        <p>Clique no botão </p>
        <button onclick="alert('Evento Onclick')">Clique Aqui</button>
    </body>
</html>
```

9.3 Evento onsubmit

O evento onsubmit é suportado apenas por formulários e disparado quando o usuário clica no botão Enviar de seus formulários. É neste evento que você fará a validação do formulário, assim como enviará o usuário para uma página, informando do sucesso/falha do envio de dados, ou outra ação que você queira definir no momento.

Sintaxe em HTML:

```
<form onsubmit="AlgumCodigoJavaScript">
```

Exemplo: onsubmit

Para observarmos o evento onsubmit iremos fazer a validação de um formulário de login que irá conter os campos nome de usuário e senha que serão verificados quando o usuário clicar no botão logar. Essa ação do usuário irá gerar um evento onsubmit o qual irá chamar a função “validacao()”.

Vamos construir o nosso formulário de login com a nossa função validacao().

```
<!DOCTYPE html>
<html lang="pt">
    <head>
        <meta charset="utf-8">
        <title>
            Pagina Principal
        </title>
        <script type="text/javascript">
            function validacao(){
                var user = "jonas";
                var password = "qwe123";
                if ((user == document.form1.usuario.value)
                    &&(password==document.form1.senha.value)){
                    window.alert("Seja bem-vindo "+user+"");
                    return true;
                }else{
                    window.alert("Usuário ou senha inválido");
                    return false;
                    }
                }
        </script>
    </head>

    <body>
        <form action="pagina1.html" name="form1" method="post" onsubmit="return validacao();">
            <center>
                <table>
                    <tr>
                        <td>Digite o nome de usuário</td>
                        <td><input type="text" name="usuario"></td>
                    </tr>
                    <tr>
                        <td>Digite a senha:</td>
                        <td><input type="password" name="senha"></td>
                    </tr>
                </table>
                <table>
                    <tr>
                        <td><input type="submit" value="Logar"></td>
                        <td><input type="reset" value="Limpar"></td>
                    </tr>
                </table>
            </center>
        </form>
    </body>
</html>
```

Vamos analisar nossa função validacao():

```
<script type="text/javascript">
    function validacao(){
        var user = "jonas";
        var password = "qwe123";
        if ((user == document.form1.usuario.value)
            &&(password==document.form1.senha.value)){
            window.alert("Seja bem-vindo "+user+"");
            return true;
        }else{
            window.alert("Usuário ou senha inválido");
            return false;
            }
    }
</script>
```

Observe que nossa função retorna **true** se o usuário e senha digitados no componente de name “usuario” e de name “senha” do formulário de name “form1”, sejam for “jonas” e “qwe123” respectivamente, caso contrário ela retorna **false.**

Esse valor retornado pela função **validacao()** irá ser salvo no evento onsubmit que caso seja **false** irá impedir que o formulário seja direcionado para a página html de nome pagina1.html , caso contrário irá permitir o redirecionamento da página.

- **Atenção**

Perceba que adicionamos a palavra reservada **return** antes da chamada de nossa função, essa palavra foi utilizada com intuito de que o evento onsubmit espere e receba um retorno da função validacao().

Evento onmouseover e onmouseout

O evento onmouseover ocorre quando o ponteiro do mouse se posiciona sobre um elemento da página HTML, já o evento onmouseout acontece o oposto, o evento é disparado quando retiramos o ponteiro do mouse de sobre um determinado elemento.

Sintaxe em HTML:

```
<elemento onmouseover="AlgumCodigoJavaScript">
<elemento onmouseout="AlgumCodigoJavaScript">
```

Exemplo:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Usando eventos no Javascript</title>

  <script type="text/javascript">
  function mOver(obj) { obj.innerHTML="Obrigado" }
  function mOut(obj) { obj.innerHTML="Passe o mouse" }
  </script>
</head>
<body>
    <div onmouseover="mOver(this)" onmouseout="mOut(this)" style="background-
    color:gray;width:120px;height:20px;padding:40px;">Passe o mouse
    </div>
</body>
</html>
```

9.4 Eventos: onfocus e onblur

O evento onfocus ocorre quando um elemento/objeto recebe o foco. Ex: quando você clica em um campo de texto para digitar algo, aquele elemento ganhou foco e nessa ação é gerado um evento do tipo onfoucs, já o onblur é o oposto o evento é gerado quando o elemento perde o foco, ou seja você tem 2 campos de textos para digitar seus dados. O primeiro você deve digitar seu nome e no outro o seu email. Após digitar o seu nome você terá que digitar o email, certo? Se você tivesse colocado um evento onBlur no campo onde digitou o nome, ao clicar no campo de e-mail um evento do tipo onBlur será gerado, pois o campo nome acaba de perder o foco na aplicação.

```
<elemento onfocus="AlgumCodigoJavaScript">
<elemento onblur="AlgumCodigoJavaScript">
```

Sintaxe em HTML:

Exemplo:

Vamos implementar dois campos nome e e-mail, os quais inicialmente estarão preenchidos com a String `"digite aqui"` quando for dado foco a um dos campos o evento onfocus será gerado e o campo de texto que ganhou foco e que estava inicialmente com uma frase se encontrará vazio para ser digitado uma informação do usuário, caso o usuário não digite nenhuma informação, por esquecimento, e retire o foco do campo de texto. O evento onblur será gerado e uma frase em vermelho irá surgir dentro do campo que perdeu o foco `"campo obrigatório"`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Usando eventos no Javascript</title>
    <script type="text/javascript">
      function alertaCampoVazio(campo){
        if (campo.value == '') {
          campo.value = 'campo obrigatório';
          campo.style.color = "red";
        } else{
            if (campo.value == 'digite aqui' || campo.value == 'campo obrigatório') {
            campo.value = '';
            campo.style.color = "black";
            };
          };
      }
    </script>
  </head>
  <body>
    <label>Nome:</label>
    <input value="digite aqui" type="text"
      onblur="alertaCampoVazio(this);"
      onfocus="alertaCampoVazio(this);">
    <label>E-Mail:</label>
    <input value="digite aqui" type="text"
      onblur="alertaCampoVazio(this);"
      onfocus="alertaCampoVazio(this);">
  </body>
</html>

```

9.5 Evento: onchange

O evento onchange é disparado sempre que o texto de um elemento foi alterado. Nós podemos usá-lo para fazer uma verificação em um valor digitado pelo usuário e informá-lo ou instruí-lo a digitar novamente. Este evento está disponível para os campos texto, áreas de texto e listas de seleção.

Sintaxe em HTML:

```
<elemento onchange="AlgumCodigoJavaScript">
```

Exemplo:

Como demonstração do uso desse evento, vamos validar um campo de senha. Iremos criar um campo para o usuário digitar sua senha e outro para que ele repita a senha digitada. O evento onchange será gerado quando o usuário modificar o conteúdo da caixa senha e fazer com que a mesma perca o foco. Quando o evento for gerado será verificado se o tamanho da senha digitada é maior ou igual a 6, caso seja verdade a sentença, a String “Senha válida” será exibida em uma caixa de diálogo alert, caso contrário será exibido a frase “Tamanho da senha inválido, digite uma senha de no mínimo 6 dígitos", o campo será limpo e ganhará o foco da aplicação novamente.

Veja abaixo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Usando eventos no Javascript</title>
    <script type="text/javascript">
      function valida(campo){
        if (campo.value.length >=6) {
          alert("Senha válida");
        } else{
          alert("A senha deve ter no mínimo 6 digitos");
          campo.value ='';
          campo.focus();
        };
      }
    </script>
  </head>
  <body>
    <label>Senha:</label>
    <input value="" type="password" onchange="valida(this);">
    <label>Repita a senha:</label>
    <input value="" type="password">
  </body>
</html>
```

9.6 Eventos: onload e onunload

O onload é um dos eventos mais importantes em JavaScript. Ele é disparado sempre que a página estiver carregada por completo. Todas as vezes que você escrever um script que só possa ser executado usando outros elementos da página, é sempre uma boa idéia usar este evento para dar início à execução do código. Por só ocorrer quando a página estiver carregada por completo, este evento evita que o seu código tente acessar algum elemento que não esteja baixado para a página ainda.

Já o evento onunload é disparado assim que usuário encerra a execução da página atual para exibir uma nova página. Ele pode ser usado para dar alguma informação, agradecer ao usuário a sua visita ou exibir uma nova página. Este evento também ocorre se você usar o botão Voltar do browser.

Sintaxe m HTML:

```
<body onload="AlgumCodigoJavaScript">
<body onunload="AlgumCodigoJavaScript">
```

Exemplo:

Vamos ver um exemplo da aplicação destes eventos. Queremos exibir uma mensagem de boas-vindas

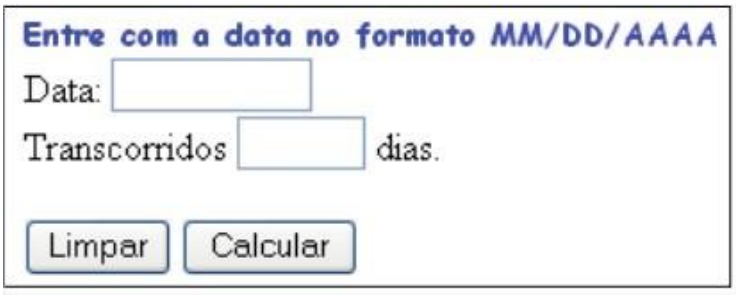
quando o visitante acessar a nossa página e outra mensagem quando ele sair da nossa página.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Usando eventos no Javascript</title>
    <script type="text/javascript">
      function saudacaoChegada(){
        alert("Seja bem vindo ao nosso site!");
      }
      function saudacaoSaida(){
        alert("Volte sempre ao nosso site!");
      }
    </script>
  </head>
  <body onload="saudacaoChegada()" onunload="saudacaoSaida()">
    <h1>Teste do onload e onunload</h1>
    <p>
      "Tinha um monte de coisas que eu queria fazer...<br>
      Eu quero ser gerente, eu quero tbm ser astronauta...<br>
      Eu quero ter minha propria loja de bolos...<br>
      Eu quero ir numa loja de rosquinhas e dizer "eu quero todas!"...<br>
      Eu quero poder viver umas 5 vezes...<br>
      Entao eu renasceria em 5 cidades diferentes...<br>
      Me encheria de coisas diferentes e deliciosas 5 vezes cada...<br>
      E entao nessas 5 vezes... eu me apaixonaria pela mesma pessoa" <br>
      <i>Inoue Orihime - Bleach</i>
    </p>
  </body>
</html>
```

Atenção! O evento onunload não é suportado pelo navegador Google Chrome ou Opera, nem por algumas versões do Mozilla Firefox.

Exercícios Propostos

Criar um contador do número de dias transcorridos desde uma determinada data, conforme formulário mostrado abaixo:



Crie um programa que apresente cinco retângulos de 100px por 200 px de cores vermelho, verde, azul amarelo e rosa. O sistema deverá então, randomicamente, o retângulo que contém um prêmio. Quando o usuário clicar em um dos retângulos, o sistema deverá então informar se aquele é o premiado.

Caso, contrário, deverá informar que o usuário errou, informando qual era o correto. (Dica: estude o método random() contido no Objeto Math)

Crie um botão em um ficheiro html, que ao ser clicado aparece um número aleatório entre 0

e 50 e 50

Crie um botão em um ficheiro html, que ao ser clicado aparece um número aleatório entre 0

Crie uma caixa de texto em html. Crie um botão em html que chama código em JavaScript

que dependendo do valor introduzido mostra uma mensagem diferente;

- Entre 0 e 10, 10 excluído, mostra “Insuficiente”;
- Entre 10 e 14, 14 excluído, mostra “Bom”;
- Maior que 14, mostra “Muito Bom”

Crie uma caixa de texto e um botão em html; ao clicar no botão, chama uma função com um parâmetro que é o valor que está dentro da caixa de texto; A função mostra mensagens de acordo com o parâmetro da caixa de texto; por exemplo, no caso de ter introduzido três, a função seria chamado com o parâmetro 3, assim, as mensagens seriam “AIA1”, “AIA 2” e “AIA 3”

10.0 Document Object Model – DOM

Para ter um domínio completo de JavaScript, é preciso que você passe pelo estudo do DOM (Document Object Model), que é o modelo dos objetos que formam a hierarquia de toda a estrutura de uma página web. Essa estrutura começa a partir da janela do navegador e vai até o último elemento da página.

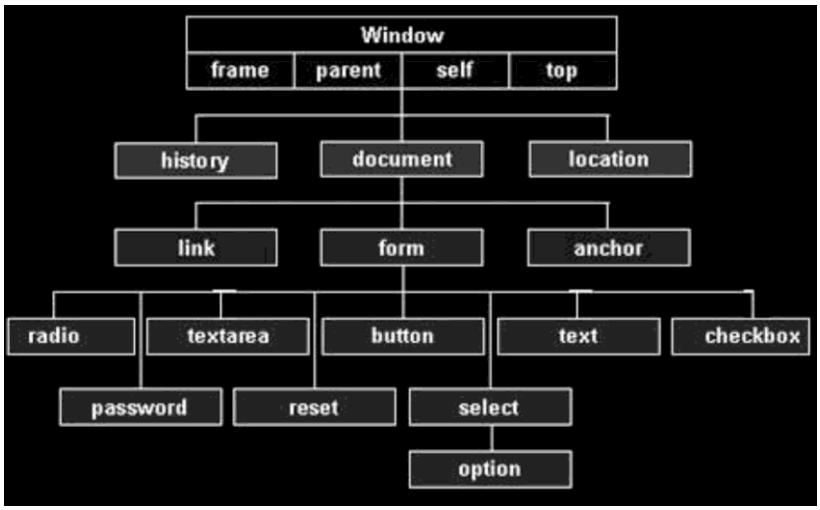
Para entendermos melhor o que é uma hierarquia de objetos, considere o seguinte exemplo.

Suponhamos que você esteja abrindo a porta de seu carro. No momento que abre a porta, se pudéssemos demonstrar em código o que você acaba de fazer, teríamos algo como o mostrado em seguida:

```
carro.porta.fechadura.aberta = true;
```

Observe que acessamos a propriedade aberta da porta, que retorna dois valores. Se estiver aberta, o valor será true e se estiver fechada, será false. Observe o ponto (.) que separa os elementos da hierarquia.

Em JavaScript as coisas não são diferentes. Observe a figura em seguida que mostra bem toda a hierarquia dos objetos em JavaScript:



Vamos agora escrever uma página HTML bem simples para que possamos ver o DOM em ação. Crie uma nova página HTML, digitando o código seguinte no seu editor favorito. Veja como a página ficará.

Lembre-se de criar as imagens que farão parte da página.


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Usando DOM</title>

  </head>
  <body align="center">
    <h3>Demonstração do DOM em JavaScript</h3>
    <img src="img/javascript_logo.png" width="154" heigth="170">
    <a href="http://ead.projetoejovem.seduc.ce.gov.br/ejovem/"></a>
    <p>Para receber  nossa newsletter difite o seu nome:</p>
    <form name="form1">
      <input type="text" name="text1">
      <input type="button" value="Enviar"><br><br>

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

E aqui está o código para a página:

Observe que a partir de agora, sempre que você criar um formulário, deverá dar nome a ele. Isso é fácil de entender. Como todos os elementos de uma página seguem o modelo de hierarquia, você deve segui- la para conseguir acessar as propriedades dos objetos. Vamos ver algum exemplo como isso é possível.

Depois de criada a página apresentada, salve-a em uma pasta. É essa página que usaremos para os nossos testes.

O primeiro script que vamos criar vai obter o título da página e exibi-lo em uma caixa de mensagem.

Abra a nossa página de testes e digite o código seguinte dentro das tags <head></head>.

```
<script type="text/javascript">
  window.alert(document.title);
</script>
```

Observe que tudo que fizemos aqui foi criar uma mensagem alert e definir que a mensagem a ser exibida será o título da página. Para acessar a propriedade title do objeto document, você só precisa usar a linha de código:

```
document.title;
```

Como este objeto está logo abaixo do objeto window na hierarquia, você pode perfeitamente acessar a propriedade acima, usando a linha de código seguinte:

```
window.document.title;
```

Por padrão, não é necessário usar o nome do objeto window sempre que for acessar o objeto document. Por esta razão nós o omitimos no script.

O nosso próximo script é um pouco mais avançado. Agora vamos criar uma função que altera o título da página em tempo de execução. Pense como uma visitante ficaria satisfeito se você exibisse o nome dele na barra de títulos do navegador.

Ainda com a nossa página de testes, digite o código abaixo na parte <body></body> da página para criarmos um botão. Para separar esse botão do outro conteúdo da página, digite algumas quebras de linhas
 para colocá-lo mais abaixo na página:

```
<form name="form1">
  <input type="text" name="text1">
  <input type="button" value="Enviar"><br><br>
  <input type="button" value="Alterar o Título da página" onclick="alterarTitulo()">
</form>
```

```
<script type="text/javascript">
  window.alert(document.title);
  function alterarTitulo () {
    window.document.title = "Seja bem vindo Samia";
  }
</script>
```

Agora, com o botão devidamente criado, digite o código seguinte na parte <head></head> da página:

Nossa página ficará dessa forma.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Usando DOM</title>
    <script type="text/javascript">
      window.alert(document.title);
      function alterarTitulo () {
        window.document.title = "Seja bem vindo Samia";
      }
    </script>
  </head>
  <body align="center">
    <h3>Demonstração do DOM em JavaScript</h3>
    <img src="img/javascript_logo.png" width="154" heigth="170">
    <a href="http://ead.projetoejovem.seduc.ce.gov.br/ejovem/"></a>
    <p>Para receber  nossa newsletter digite o seu nome:</p>
    <form name="form1">
      <input type="text" name="text1">
      <input type="button" value="Enviar"><br><br>
      <input type="button" value="Alterar o Título da página" onclick="alterarTitulo()">
    </form>
  </body>
</html>
```

Salve a página e execute.

Você vai perceber que, assim que clica no botão, ele chama a função alterarTitulo() e o título da página é alterado. Legal, não?

Agora, que tal colocar o nome do visitante na barra de títulos? Vamos ao nosso script:

O código seguinte pede que o visitante digite o seu nome e depois o exibe na barra de títulos juntamente com o título da página. Digite o código seguinte na parte <body></body> da página de teste:

```
<script type="text/javascript">
  var nome = window.prompt("Qual é o seu nome?", "Seu nome");
  var titulo = document.title + " - Visitante: " + nome;
  document.title = titulo;
</script>
```

Que tal usar isso em alguma página? Tenho certeza de que os seus visitantes ficarão deslumbrados.

Vamos entender um pouco como o código foi feito.

Primeiro nós criamos uma variável chamada nome e atribuímos a ela uma mensagem prompt, perguntando o nome do visitante. Criamos também uma variável chamada ”titulo” que receberá o texto do título do documento mais o valor da variável “nome”, que será informada pelo visitante. A partir daí é só alterar o título por meio da linha de código:

```
document.title = titulo;
```

Fácil, não?

Fico imaginando o que seria de nós, programadores em JavaScript, se não pudéssemos acessar os objetos por meio da hierarquia do DOM do JavaScript.

O script seguinte mostra como acessar uma caixa de texto e pegar o valor que foi digitado pelo usuário. Observe que sempre que quiser realizar tal tarefa, terá que dar nomes tanto ao formulário quanto aos seus elementos. Se você ainda não tem um conhecimento muito aprofundado de HTML, veja como dar nomes aos elementos:

```
<form name="meu_form">
<input type="text" name="texto1">
```

Quando você dá nomes aos elementos, pode acessá-los e manipular suas propriedades do jeito que quiser. Você pode dar qualquer nome aos seus elementos. Apenas tente dar nomes sugestivos que faça com que você se lembre mais facilmente na hora de acessar cada um deles.

Voltando à nossa página de teste, você pode ver que temos um formulário chamado form1, uma caixa de texto chamada text1 e um botão. Observe que aqui não demos um nome ao botão, mas se quisermos acessar suas propriedades (faremos isso no próximo script), temos que dar um nome a ele.

O script que vamos escrever agora exibe uma mensagem assim que o usuário digita o nome na caixa de texto e clica no botão Enviar.

O texto a ser exibido usa o valor digitado na caixa de texto. Vamos ao código: Digite este código na parte <head></head> de sua página:

```
function enviar () {
  var nome = document.form1.text1.value;
  window.alert("Você digitou: " + nome);
}
```

Agora, precisamos chamar esta função a partir do botão Enviar. Para tanto, vá até o botão e digite a seguinte linha de código:

```
onclick="enviar()"
```

O seu código completo para o botão deve ficar assim:

```
<input type="button" value="Alterar o Título da página" onclick="alterarTitulo()">
```

Salve a página, execute, digite o seu nome na caixa de texto e clique o botão Enviar. Se você realizou todos os procedimentos corretamente, verá uma caixa de mensagem exibindo o texto Você digitou "seu nome". Como nos exemplos anteriores, pratique bastante alterando valores e mixando estes scripts com as técnicas aprendidas anteriormente.

Exercícios Propostos

Crie uma função que seja capaz de receber a informação de 2 campos de texto e em seguida mostre no HTML os valores digitados.

No mesmo formulário melhore a função verificando se existe ou não algo digitado em um dos campos, se não tiver nada digitado, o html deverá mostrar uma mensagem dizendo campo vazio.

Crie um formulário para que irá receber 3 valores, converta-os para valores numéricos usando Objeto Number() e imprima a média dos valores digitados.

Crie um formulário com um campo de texto para uma nota e um combobox que terá as matérias: Física, matemática e português. Mostrei abaixo qual nota foi digitada e qual curso foi selecionado.

11.0 Introdução ajQuery

O FrameWork jQuery nada mais é que uma um conjunto de bibliotecas escritas em JavaScript,

contendo funções que tratam de efeitos interativos em processos Web. O jQuery veio para facilitar os efeitos do JavaScript, realizando em poucas linhas uma vastas possibilidades de efeitos. Para ter um bom aprendizado, será importante conhecer bem o JavaScript, especialmente modelo de Objeto de Documento ou seja **(DOM)** e **Eventos.** Pois podemos ter interações com todos os elementos da estrutura básica de um site. Nesse curso iremos ver **alguns** comandos simples do jQuery, em seguida, nos focaremos na biblioteca **jQueryUI,** utilizado para abstrair a programação complexa de baixo nível em interações WEB/usuário aumentando a produtividade, funções e beleza do site.

Um exemplo de sua funcionalide:

Enquanto em Java Script escrevíamos: **document.getElementById(“elemento”)**
No jQuery só precisamos : **$(“#elemento”)**

11.1 Instalação e configuração

Para realizar a instalação e configuração, primeiramente devemos acessar o site oficial da jQuery: “http://code.jquery.com/” E acessar o link de download. Você deverá copiar o conteúdo de texto e salvar no seu computador como **“jquery.js”**. Como iremos trabalhar na versão: 2.1.4, o link direto que contém o a jquery é : **“http://code.jquery.com/jquery-2.1.4.min.js”.** Assim que o arquivo tiver salvo, adicione-o dentro do seu projeto, de preferência em uma pasta para separá-la dos demais elementos do seu site .

Pronto, falta apenas importar a biblioteca para sua página Web, para isso, basta importar o jQuery na sua

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

página WEB digitando o seguinte comando na estrutura <head>:

Perceba que fizemos esse mesmo procedimento em capítulos anteriores para importação dos nossos projetos em JavaScript. A diferença que copiamos um projeto já feito pelo grupo da jQuery que importamos no nosso site. Podemos também criar o nosso projeto e importá-lo abaixo, unindo a nossa biblioteca com a do jQuery. Veja no exemplo abaixo;

```
<script type="text/javascript" src="js/anima.js"></script>
<script type="text/javascript" src="js/jquery.js"></script>
```

Obs.: Se por acaso precisarmos migrar um site para um outro projeto que tiver recursos do jQuery implementados, precisamos exportar a biblioteca junto ou linkar o arquivo externamente alterando o caminho src para um endereço da página web onde encontra-se a biblioteca jQuery, no nosso caso: **"code.jquery.com/jquery-2.1.4.min.js".**

11.2 Primeiros Passos

Para inicializar as funções do JavaScript e jQuery precisamos primeiro criar um evento de inicialização do documento DOM. Inicialmente vamos gerar um evento do tipo "ready", ou seja, quando o documento raiz da nossa página estiver pronto e iniciado veja a imagem abaixo.

```
<script type="text/javascript">
  $(document).ready(
  function () {
    alert("Função principal na inicialização da página");
  });
</script>
```

O termo "$(document) " indica que estamos acessando o documento DOM como um todo. O termo "ready" indica que quando o DOM estiver pronto, a página irá inicializar um evento. O evento está indicado
por "function()". Dentro dessa "função principal" podemos realizar nossas animações, criar outros eventos, etc.

```
<script>
$(document).ready(
    function(){
        $("a").click(
            function(){
                alert("o botão foi pressionado");
            }
        );
    });
</script>
```

No exemplo acima, geramos uma mensagem ao entrar na página, iremos agora gerar um, ao evento, ao clique de um botão obedecendo o mesmo padrão.
Usando o mesmo raciocínio, ativamos um evento de alerta, quando um botão com referência de link "a" for clicado. Assim podemos manipular qualquer elemento que esteja presente na nossa página WEB.

Estrutura

HTML para script acima:

```
<body>
    Clique no botão: <br />
    <a href=""> Primeiro Evento </a><br />
</body>
```

Exercício Rápido: Como seria ativar um botão ao clicar ou ao selecionar uma div?

Usando o método do exemplo acima, criamos um mesmo evento caso clicássemos em qualquer elemento de link "a". Porém como poderíamos criar eventos isolados por referência de elemento? Para isso, podemos usar "#id" ou ".class" conteúdo esse visto no nosso curso de HTML/CSS. Iremos agora gerar um evento isoladamente para um botão, veja no exemplo abaixo.

```
<script>
$(document).ready(
    function(){
        $("#evento1").click(
            function(){
                alert("Evento 1 ativo");
            }
        );

        $("#evento2").click(
            function(){
                alert("Evento 2 ativo");
            }
        );

    });
</script>
```

Perceba que agora não iremos mais ativar um evento ao clicar em uma tag "a" qualquer, podemos direcionar um evento para um elemento específico do nosso HTML usando "#id". Sabemos que um ID só poderá ser usado em um elemento HTML por página, então podemos usar esse parâmetro para gerar eventos em elementos individuais.

Estrutura HTML para script acima:

```
    <body>
        Clique no botão: <br />
        <a id="evento1" href=""> Primeiro Evento </a><br />
        <a id="evento2" href=""> Segundo Evento </a>
    </body>
</html>
```

Podemos também criar funções separados do HTML, da mesma forma que fizermos no curso de JavaScript, faça um teste, adapte o código acima com funções em um aquivo chamado “anima.js” e chame-o dentro do seu código.

```
    <script>
        $(document).ready(
        function(){
            $("#content a").click(
            function(){
                alert("Evento acionado");
            });
        });
    </script>
</head>
<body>

    <div id="content">
        <a href=""> Pronto Para Evento JS</a>
    </div>
        <a href=""> Sem evento JS</a>
</body>
```

Se você quiser acessar um elemento que está dentro de um outro elemento identificado por algum “id”, “class” ou “tag”, podemos colocar o caminho
completo do elemento que queremos acessar da seguinte forma:

Perceba que um dos links “<a>” está dentro da div “content”. Então geramos um evento para ser acionado apenas no link que está dentro dessa div adicionando o caminho :

$(“#content a”).evento

Veremos no próximo tópico outras formas de acesso a elementos HTML pelo jQuery:

Exercício Rápido: Como seria ativar esse mesmo evento levando em consideração que a camada “#content” estaria dentro de uma outra camada chamada “fundo”?

11.2.1 Métodos de acesso a Elementos HTML

Mostraremos algumas formas de selecionar um elemento dentro da sua estrutura HTML. Conhecer os exemplos abaixo é bastante importante, pois em nossos algoritmos estaremos sempre acessando e alterando valores das nossas páginas WEB. Estaremos usando alguns desses seletores no decorrer do nosso conteúdo. Seletores:

Seletor	Descrição
$(this)	Seleciona o elemento referendado por ela mesma.

$('*')	Seleciona todos os elementos do documento HTML
$(' div ')	Seleciona todos os elementos <DIV> poderá ser usado para outros elementos. Ex: “img”, “a” , “table”.
$(' .curso ')	Seleciona todos os elementos cuja a classe seja “curso”.
$(' $camada ')	Seleciona um único elemento de id = “camada”.
$('$camada img')	Seleciona todas as imagens que estiver dentro da camada (exemplo).

Pseudo-Seletores:

Seletor	Descrição	Exemplo
:first	Seleciona por um primeiro item	$(“ul li:first”)
:last	Seleciona por um ultimo item	$(“ul li:last”)
:not	Ignora um item selecionado	$(“ul li:not(:last)”)
:contains	Seleciona um item que contenha texto indicado.	$(“p:contains(fab)”)
:empty	Seleciona itens vazio	$(“p:empty”)
:visible	Seleciona itens visiveis	$(“div:visible”)

Pseudo-Seletores de formulários:

Seletor	Descrição	Exemplo
:text	Seleciona campos do time text	$(“:text”)
:hidden	Seleciona campos invisíveis	$(“:hidden”)
:file	Seleciona arquivos de um formulários	$(“:file”)
:button	Seleciona botões do tipo “button” em um formulário	$(“:button”)
:checkbox	Seleciona um checkbox de formulário	$(“:checkbox”)
:image	Seleciona campo/Inagem de um formulário	$(“:image”)
:password	Seleciona um campo de texto do tipo Senha de um formulário	$(“:password”)
:radio	Seleciona uma caixa de texto do tipo Radio	$(“:radio”)
:submit	Seleciona um botão do tipo submit de um formulário.	$(“:submit”)

:reset	Seleciona um botão do tipo reset de um formulário.	$(“:reset”)

Seletores de Formulários:

Seletor	Descrição
$('input [name = “bot”]')	Seleciona todos os elementos de formulário com o nome “bot”.
$('input [name != “curso”]')	Seleciona todos os elementos de formulário que contenha um valor diferente de “curso”.

12.0 BibliotecajQuery

Neste capítulo estudaremos algumas funções da biblioteca jQuery, não estudaremos toda a biblioteca,

pois são muitas funções, o objetivo do nosso curso é mostrar através de algumas funções o uso prático dessa ferramenta, se você pretende se especializar nesse framework indicamos esse link: "http://api.jquery.com/". Este link contém uma documentação completa de todas as funções do jQuery com exemplos práticos do jQuery para as mais diferentes situações.

12.1 Funções de Estilo

Com a biblioteca jQuery, Podemos alterar dinamicamente os estilos de elementos HTML de forma simples, podendo adicionar ou alterar estilos através dos comandos "css()", "add()" , "addClass()" , "removeClass()" e "toggleClass()". Veremos em exemplos suas diferenças, como podemos utilizar esses recursos para cada caso utilizando também recursos aprendidos no módulo anterior de JavaScript.

12.2 Função .css()

Com esse comando podemos adicionar um estilo CSS mesmo que o elemento html não tenha estilo pronto para ela, vejamos como podemos aplicar isso.

$("elemento").css("Atributo CSS", "valor")

```
    <script>
        $(document).ready(
        function(){
            $("#bot").click(
            function(){
                $("#texto").css("color","red");
            });
            }
        );
    </script>
</head>
<body>
    <p id="texto"> Texto a ser modificado uma cor </p>
    <button id="bot"> Aletar estilo </button>
</body>
```

No exemplo abaixo, temos um botão que aciona um evento responsável por criar um estilo ao texto Html veja:

Este código mostra como alterar a cor de um texto usando o jQuery. Criamos um evento de clique em um botão chamado "bot" que será responsável por alterar a cor do texto, perceba a cor do texto que será

alterada está em um parágrafo que tem como seu id = “texto”. Alteramos o conteúdo do texto para cor vermelha juntamente no na função .css(“color”, “red”); . Podemos alterar qualquer CSS de qualquer elemento usando essa função.

Podemos também guardar em uma variável JS, informações de algum atributo CSS já criado usando:

```
var corAtual = $(“elemento”).css(“Atributo CSS”);
```

Exercício Rápido: Como seria alterar a cor de plano de fundo de um campo de texto de formulário ao selecionar o clique?

12.3 Função .add()

Com este comando podemos adicionar estilo a vários elementos distintos com um mesmo comando, podemos também definir os estilos mais genéricos, e específicos entre os elementos envolvidos no relacionamento.

No exemplo abaixo, temos um exemplo de uma tabela editada com recursos de estilo “add()”. Iremos posteriormente analisar o código.

```
    <script>
        $(document).ready(
        function(){
            $("th").css("color","navy")
                        .add("#tabelaEdit td")
                        .css("background-color", "#DDDDDD");
        });
    </script>
</head>
<body>
    <table id="tabelaEdit" border="1">
        <thead>
            <tr>
                <th>Nome</th>
                <th>e-mail</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Fabricio</td>
                <td>emailFabricio@gmail.com</td>
            </tr>
            <tr>
                <td>Rosal</td>
                <td>emailRosal@gmail.com</td>
            </tr>
        </tbody>
    </table>
</body>
```

Se olharmos o código JS de trás para frente, podemos perceber que incluímos em todas as nossas células “td” que estão dentro da tabela com id = “tabelaEdit” um plano de fundo de cor cinza. Continuando a leitura, as tags “th” recebem um estilo de cor de texto com nome “navy”, porém as tags “th” também irão “herdar” as características de estilos que foram atribuídas à tag “td”. Portanto o plano de fundo da tag “th” também será preenchido como cinza em seu plano de fundo.

Veja como ficou a nossa tabela.

Nome	e-mail
Fabricio	emailFabricio@gmail.com
Rosal	emailRosal@gmail.com

Exercício Rápido: Como seria aplicar esse mesmo conceito em uma lista HTML?

13.1.1. Função .addClasse() e removeClass().

Estas funções são responsáveis por alterar um estilo estruturado de CSS de um determinado elemento html substituindo um estilo por outro. Veja sua Sintaxe:.

```
$("elemento").addClass(".classNova");
```

E para remover o estilo da classe adicionada temos:

```
$("elemento").removeClass(".classNova");
```

Iremos criar um algoritmo para alterar o estilo de uma DIV utilizando o evento hover, ou seja, quando o mouse estiver selecionado sobre a camada. Para isso iremos inicialmente criar 2 classes de estilo para uma

```
.estilo1{
    width: 200px;
    height: 100px;
    background-color: #999900;
    margin: 10px;
    color: yellow;

}
.estilo2{
    width: 200px;
    height: 100px;
    background-color: blue;
    margin: 10px;
    color: white;
}
```

```
    <script>
        $(document).ready(
        function(){
            $(".estilo1").hover(
            function(){
                $(this).addClass("estilo2");
            },function(){
                $(this).removeClass("estilo2");
            });
        });
    </script>
</head>
<body>
    <div class="estilo1">
        Texto Teste.
    </div>
```

camada e usar os comandos da seguinte forma:

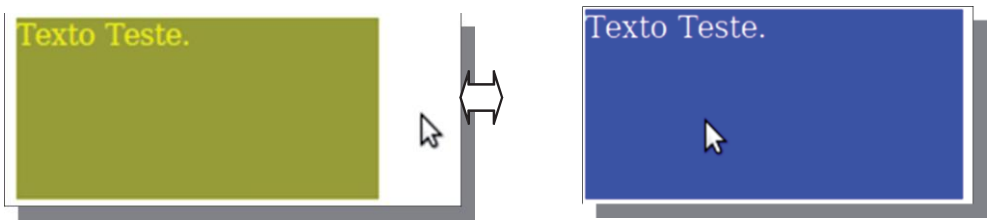
Vamos analisar cada parte do algorítimo:

$(".escilo1").hover()- Com esse comando, iremos chamar um evento ao selecionar o mouser por cima do elemento. No nosso caso na DIV que estiver com estilo ".estilo1" associado.

$(this).addClass("estilo2") - Com esse comando estamos querendo alterar a própria camada que foi selecionada, indicado pelo termo **"this",** para o **".estilo2".**

$(this).removeClass("estilo2") - Quando o mouser for retirado da camada, o queremos retirar o estilo que foi incluído anteriormente na camada. Voltando ao seu estado original.

Veja como ficaria essa operação no browser:



Exercício Rápido: Crie 2 camadas, onde cada camada seja responsável por alterar temporariamente o estilo da outra camada.(camada 1 altera estilo da camada 2 e vice-versa)

Podemos alterar não só as Div, mas também qualquer elemento HTML

12.4 Função toggleClass()

Com essa função podemos montar de forma mais simples a animação do tópico passado. podemos modificar um estilo de um elemento e quando esse elemento for ativado novamente, a função irá voltar para o seu estado original. Veja sua Sintaxe:.

```
$(“elemento”).toggleClass(“.classNova”);
```

Como será a modificação do nosso código.

```
<script>
    $(document).ready(
    function(){
        $(".estilo1").hover(
        function(){
            $(this).toggleClass("estilo2");
        });
    });
</script>
```

12.5 Funções de visibilidade

Com a biblioteca jQuery, podemos alterar dinamicamente a visibilidade de elementos HTML de forma simples, podendo mostrar ou ocultar os elementos através dos comandos “show()”, “hide()” , “toggle()”.Estas funções da Biblioteca jQuery são referentes ao modo de visibilidade, sendo possível alterar a visão de qualquer elemento da nossa estrutura HTML dinamicamente. Iremos ver nos próximos tópicos como utilizá-los.

12.6 Funções show() e hide()

Podemos mostrar um determinado elemento HTML através do comando “show()” ou esconder-los usando o comando “hide()” de acordo com as nossas necessidades. Veremos a sintaxe:

Para ocultar um elemento dinamicamente por um evento:

```
$(“elemento”).hide()
```

Para mostrar um elemento dinamicamente por um evento:

```
$(“elemento”).show()
```

As 2 funções podem receber como parâmetro uma String com 2 tipos de valores.

Fast - Tornará oculto / visível um elemento com o efeito de transição de forma rápida. Slow - Tornará oculto / visível um Elemento com o efeito de transição de forma lenta. Sintaxe com Parâmetros:

```
$(“elemento”).show(“fast”) / $(“elemento”).show(“slow”)
$(“elemento”).hide(“fast”) / $(“elemento”).hide(“slow”)
```

Se quisermos editar o tempo de duração do evento, podemos também atribuir um valor em milisegundos referente ao tempo da animação:

```
$(“elemento”).show(2000);
```

Para esse exemplo iremos mostrar uma camada em um tempo de 2000 milissegundos, ou seja,2 Segundos.

Obs.: existe um estilo CSS chamado “display” Quando ele está atribuido incialmente como “none”, queremos dizer que inicialmente o elemento que tiver esse atributo, inicialmente estará oculto, sendo assim podemos torná-los visível através do show() e tornar oculto novamente através do hide();

Vamos analisar um exemplo prático na prima página, mostrando como mostrar ou ocultar uma pequena

```
    <script>
        $(document).ready(
        function(){
            $("#bot").click(
            function(){
                $("p").hide();
                $("#divFormulario").show();
            });

            $("#voltar").click(
            function(){
                $("p").show();
                $("#divFormulario").hide();
            });
        });
    </script>
</head>
<body>
    <p>
        Deseja Responder nossa enquete?<br />
    </p>
     <button id="bot">Responder</button>


    <div id="divFormulario">
        Quantas horas de estudo você realiza por dia ?
        <form name="form1" action="#" method="GET">
            <input type="radio" name="escHoras" value="hora1" />
            Pelo menos 1 hora<br />
            <input type="radio" name="escHoras" value="hora2" />
            Pelo menos 2 horas<br />
            <input type="radio" name="escHoras" value="hora3" />
            Pelo menos 3 horas<br />
            <input type="radio" name="escHoras" value="horaMais" />
            Mais de 3 horas<br />
            <input type="submit" value="Enviar" name="botao"/>
            <button id="voltar"> Voltar </button>
        </form>
    </div>
</body>
```

enquete que poderá está dentro do seu site:

Exercício Rápido: Crie 2 formulários, onde o segundo só aparecerá quando o usuário terminar de preencher o primeiro formulário.

A camada “idFormulario” estará inicialmente com “display : none;” em seu CSS. Assim ela estará oculta incialmente. O Botão chamado “#bot” realizará um evento ao clique do botão que será responsável por esconder o paragrafo de texto e mostrar a camada “idFuncionário” que é justamente a camada que contém a enquete. Dentro da camada enquete podemos perceber que temos um botão chamado “voltar” ele fará o inverso, irá ocultar a enquete e tornará visível novamente o texto inicial que está dentro do parágrafo.

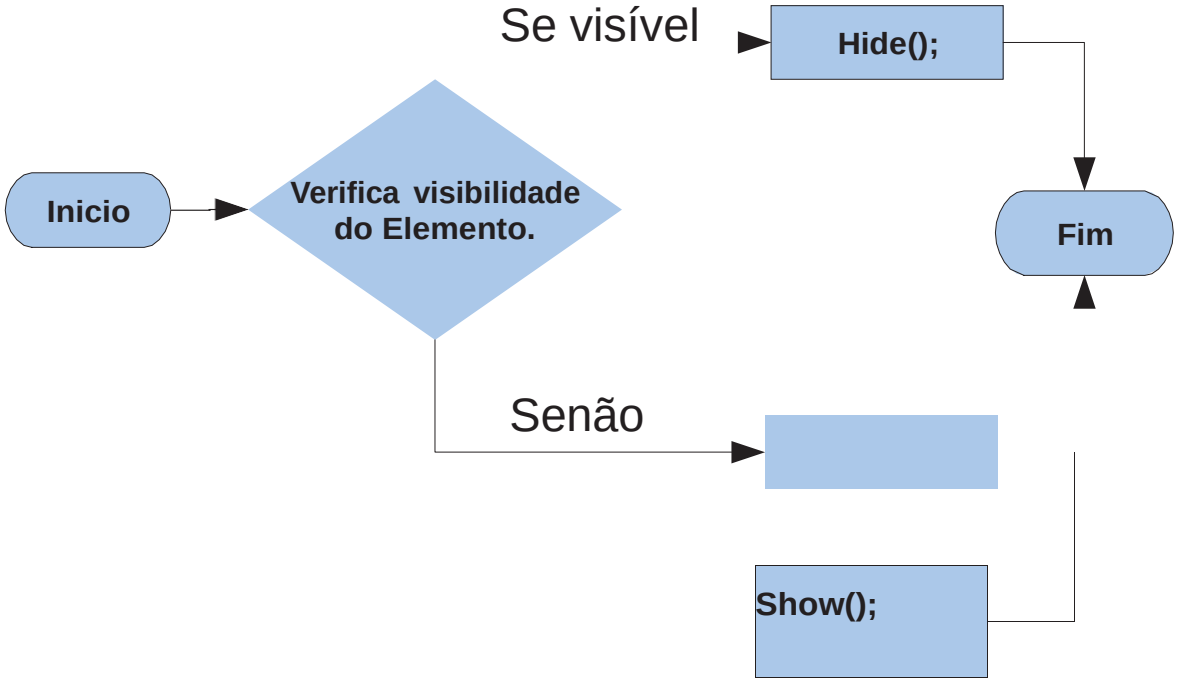
Perceba que não colocamos nenhum parâmetros na função: show() ou hide(). Veja o que acontecerá se colocarmos os parâmetros aprendidos nesse tópico.

12.6.1 Função Toggle()

Esta função, realiza o mesmo resultado das funções, show() e hide() estudados no tópico anterior.

Porém ele tem uma pequena implementação de algoritmo que permite analisar se o elemento está visível ou

não, alterando dinamicamente o seu estado de visibilidade. Veja no Fluxograma abaixo seu processo.



Veja sua Sintaxe:

```
$("elemento").toggle();
```

Obs.: Esta função permite os mesmos parâmetros aprendidos nas funções "show()" e "hide()" estudados do tópico anterior.

Vejamos um exemplo abaixo usando o "toggle();" levando em consideração a mesma estrutura HTML do tópico anterior.

```
<script>
    ocult = true;
    $(document).ready(
    function(){
        $("#bot").click(
        function(){
            $("p").toggle("slow");
            $("#divFormulario").toggle("slow");
            if(ocult){
                document.getElementById('bot').innerHTML = 'Ocultar';
                ocult = false;
            }else{
                document.getElementById('bot').innerHTML = 'Responder';
                ocult = true;
            }
        });
    });
</script>
```

Perceba que com apenas um botão botemos mostrar e ocultar vários conteúdos simultaneamente

12.6.2 Função delay()

Essa função poderá ser combinada com outras funções jQuery, ela ira programar cada evento com um tempo de espera estimado em milissegundos como parâmetro. Iremos mostrar a seguir a sua sintaxe e um exemplo de sua aplicação:

$(“elemento”).**delay(tempo)**.outrasFunções();

Iremos mostrar 2 eventos que poderá ser disparado através do clique, um deles terá um elemento com delay entre o título e o conteúdo de texto, o outro não terá, vejamos a diferença:

```
    <script>
        $(document).ready(
        function(){
            $("#b1").click(
            function(){
                $('div').add('p').hide();
                $('#conteudo1').show('slow');
                $('#conteudo1 p').show('slow');
            }
        );
            $("#b2").click(
            function(){
                $('div').add('p').hide();
                $('#conteudo2').show('slow');
                $('#conteudo2 p').delay(1500).show('slow');
            }
        );
        });
    </script>
</head>
<body>
        <button id="b1">Conteudo 1</button>
        <button id="b2">Conteudo 2</button>


    <div id="conteudo1" style="display: none">
        <h2>Titulo1</h2>
        <p>Texto para conteúdo 1</p>
    </div>


    <div id="conteudo2" style="display: none">
        <h2>Titulo2</h2>
        <p>Texto para conteúdo 2</p>
    </div>
</body>
```

Perceba que todos os eventos irão inicializar ao acionar o botão “b2”. Porém o conteúdo do paragrafo da camada div “conteudo2” irá esperar 1.5 segundos até começar seu efeito de “fadeIn()”;

12.7 Funções de Opacidade

Com a biblioteca jQuery, podemos controlar dinamicamente a opacidade de elementos HTML de forma simples, podendo tornar completamente opaco ou invisível, também podemos controlar o nível de opacidade de um elemento através dos comandos “fadeIn()”, “fadeOut()” , “fadeTo()” , e “fadeToggle()”. Veremos nos próximos tópicos como essas funções funcionam

12.7.1 Funções FadeIn() e Fadeout()

A função fadeIn() faz tornar visível um elemento HTML, enquanto fadeOut() torna invisível. Veja sua sintexe abaixo:

Para tornar visível:

$(“elemento”).**fadeIn();**

Para tornar invisível;

$(“elemento”).**fadeOut();**

Podemos também controlar a velocidade que esses elementos poderão aparecer ou sumir desta forma:

Obs.: Podemos também controlar o tempo dos efeitos de opacidade colocando um parametro “**slow”** ou **“fast”** ou atribuir valores de parametro em milisegundos ao invés de um texto, da mesma forma que fazemos para os comandos **show() e hide()** estudado nos tópicos anteriores.

Veja um exemplo abaixo usando as funções aprendidas:

```
    <script>
        $(document).ready(
        function(){
            $('input[name="bot1"]').click(
            function(){
                $('img').fadeOut('slow');
                $("#img1").fadeIn("slow");
            });
             $('input[name="bot2"]').click(
            function(){
                $('img').fadeOut('slow');
                $("#img2").fadeIn("slow");
            });
        });
    </script>
</head>
<body>
    <input type="button" name="bot1" value="Anterior" />
    <input type="button" name="bot2" value="Proximo" /><br />
    <img id="img1" src="img/img1.jpg"
        style="position: absolute" />
    <img id="img2" src="img/img2.jpg"
        style="display: none;position: absolute"/>
</body>
```

Neste exemplo temos 2 imagens que foram colocadas no nosso site. Existem 2 botões que são responsáveis por fazer um efeito de transição entre as imagens inseridas, perceba que para cada botão,

enquanto uma imagem desaparece (fadeOut) uma outra vai aparecendo (fadeIn). Veja:

12.7.2 FadeTo();

Esta função é responsável por controlar o nível de opacidade de um elemento HTML, veja sua sintaxe.

$("elemento").**fadeTo(velocidade, opacidade);**

onde:

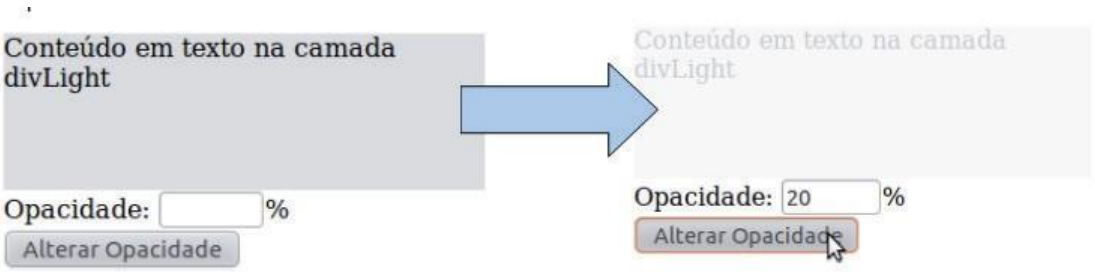
Velocidade = Velocidade do efeito de transição da opacidade antiga com a novo valor de opacidade poderá ser numérico em milissegundos ou uma String "slow" e "fast".

Opacidade = nível de opacidade do elemento, poderá variar de 0(zero) até 1(100%) onde os valores intermediários seriam o nível da opacidade. Exemplo: 0.5 = (50% de opacidade).

```
    <script>
        $(document).ready(
        function(){
            $('input[name="bot"]').click(
            function(){
                var valor = $('input[name="opacidade"]').val();
                valor = valor/100;
             $(".divLight").fadeTo("slow",valor);
            });
        });
    </script>
</head>
<body>
    <div class="divLight">
        Conteúdo em texto na camada divLight
    </div>
    Opacidade:
    <input type="text" name="opacidade" size="5"/>%<br />
    <input type="button" name="bot" value="Alterar Opacidade" />
</body>
```

Abaixo temos um exemplo que muda a opacidade de uma camada ativado por um botão que verifica o valor de uma caixa de texto e realiza a animação de mudança de opacidade.

Usamos também um novo Método para receber valores em um formulário usando jQuery, através do comando: (“elemento input”).val(). Em JavaScript, usávamos outros métodos, não fará diferença para esse exemplo, nos 2 casos podemos receber valores de campos HTML. Temos a seguir o seguinte resultado:



12.7.3 FadeToggle()

Essa função é responsável por alterar o modo de visibilidade de um elemento HTML voltando ao seu estado inicial quando ativado novamente, ele segue o mesmo raciocínio das funções que contém “Toggle” nos tópicos anteriores, alterando apenas a maneira como é feita a animação. Veja sua sintaxe:

$(“elemento”).**fadeToggle(velocidade);**

Velocidade = Velocidade do efeito de transição da opacidade antiga com a novo valor de opacidade poderá ser numérico em milissegundos ou uma String “slow” e “fast”. Esse parâmetro é opcional. Veja um Exemplo:

```
    <script>
        $(document).ready(
        function(){
            $("a").hover(
            function(){
                $("#resposta").fadeToggle();
            });
        });
    </script>
</head>
<body>

    <p> Qual a função básica do Jquery <a href="">Ver resposta</a> </p>
    <p id="resposta" style="display: none">
        jQuery foi feito para tornar mais simples a navegação<br />
        do documento HTML, a seleção de elementos DOM e <br />
        criar animações </p>
</body>
```

Resultado:

Exercício rápido: Como seria por uma legenda com uma camada semitransparente usando fade por cima das imagens em efeito?

13.0 Biblioteca jQueryUI

A jQuery UI proporciona abstrações de baixo nível de interação, animação e avançados efeitos

especiais de alto nível, construída em cima do jQuery JavaScript Library, que pode ser utilizada para construir aplicações web altamente interativas.

13.1 Instalação e configuração

A instalação e configuração é feita de forma semelhante a que fizemos com jQuery, vamos para o site Oficial do projeto jQuery UI : http://jqueryui.com/ . Onde iremos baixar um pacote para a instalação. Iremos trabalhar com a versão **jquery-ui-1.11.4.custom.**

O jQuery UI nos disponibiliza um pacote contendo um jQuery e jQuery UI, junto com uma pasta chamada CSS. Essa página também deverá ser importada no seu projeto, pois alguns recursos utilizando de estilos pré-definidos. Acesse o site da jQuery UI e você poderá receber outros pacotes com outros efeitos, por padrão usaremos o estilo **"ui-lightness"** pacote esse que você escolhe no momento do download do pacote. Você poderá acessar o CSS modificando configurações existentes ou criando um próprio estilo para você interagindo com seu jQuery.

O link de importação deverá ser da seguinte forma:

```
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript" src="js/jquery-ui.min.js"></script>
<script type="text/javascript" src="js/jquery-ui.js"></script>
<link rel="stylesheet" type="text/css" href="css/jquery-ui.min.css">
```

Pronto, agora além de melhorarmos algumas funções existentes do pacote jQuery já estudados, estamos também implementando novas funções da biblioteca jQuery UI. Estudaremos nesse curso algumas funções dessa biblioteca, recomendamos que veja a documentação do site oficial da jQuery UI para estudos complementares.

13.2 Datepicker.

Essa função é responsável por mostrar um calendário para o usuário, as informações do dia selecionado pelo usuário, poderá diretamente ser alterado em uma caixa de texto, oferecendo estilo e poupando o usuário de ter que digitar uma data manualmente.

Sua Sintaxe:

```
$("elemento").datepicker();
```

Elemento – Elemento HTML poderá ser responsável por representar o retorno da data escolhida. Veja um exemplo:

```
<html>
  <head>
    <title></title>
    <meta charset="UTF-8">
    <script type="text/javascript" src="js/jquery.js"></script>
    <script type="text/javascript" src="js/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/jquery-ui.js"></script>
    <link rel="stylesheet" type="text/css" href="css/jquery-ui.min.css">
    <script type="text/javascript">
      $(document).ready(
        function () {
          $("#data").datepicker();
        });
    </script>
  </head>
  <body>
    Faça seu agendamento: <br>
    <input type="text" id="data" size="10">
    <button>Enviar</button>
  </body>
</html>
```

Temos então o resultado a seguir no navegador:

13.3 Accordion

Essa função é responsável por segmentar conteúdos de forma mais elegante, podemos mostrar um conteúdo selecionado ocultando os outros conteúdos que tiverem dentro dessa camada. Sua Sintaxe e descrita da seguinte forma:

```
$("elemento").accordion();
```

Onde esse elemento descrito na sintaxe poderá ser uma ID de uma camada. Para organizar o conteúdo interno, devemos estruturar da seguinte forma:

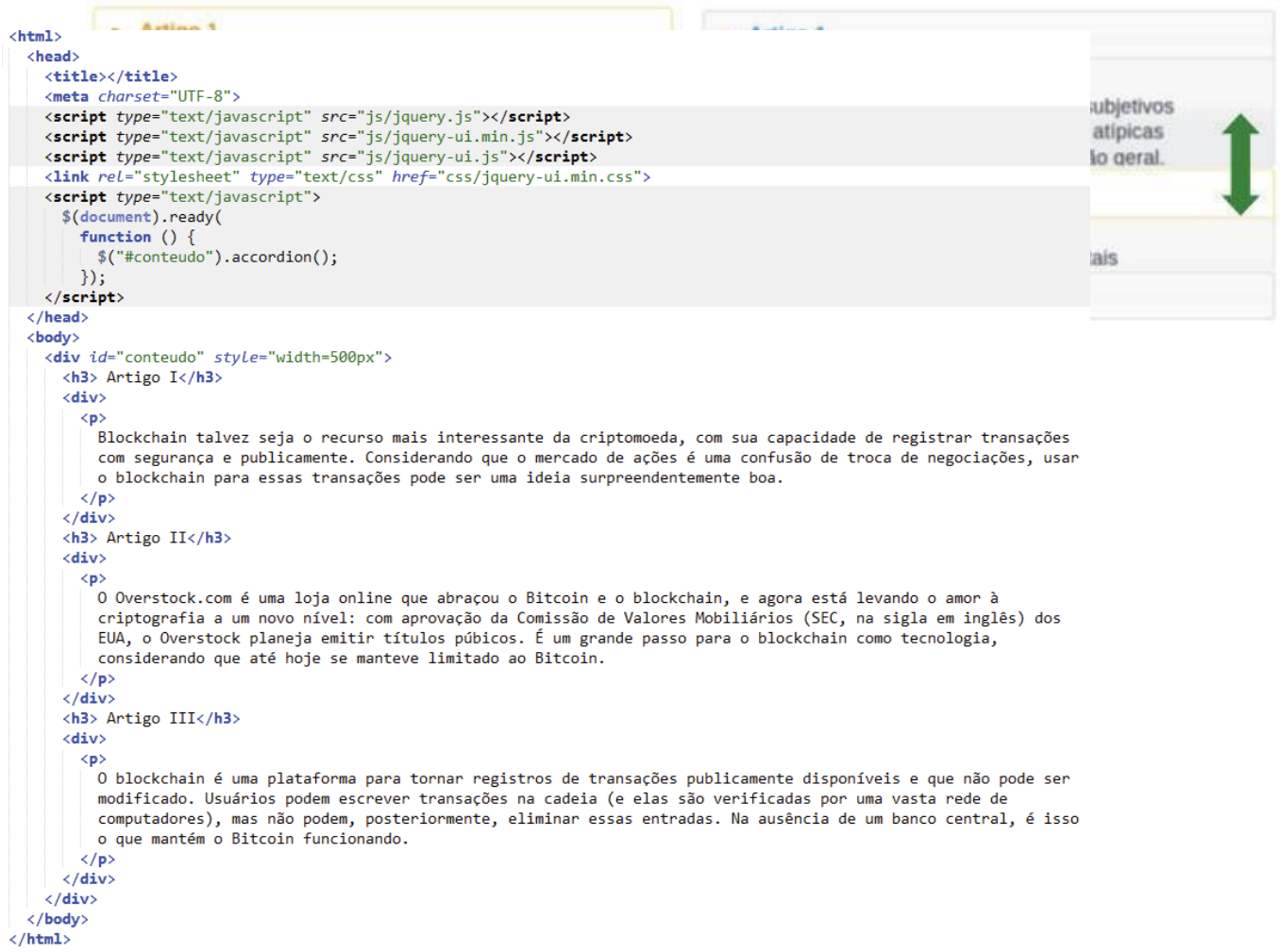
<H3>: Dentro da camada indicada com **"accordion()"** irá representar o título de cada camada de conteúdo.

** :** as Camadas internas à camada indicada pelo **"accordion()"** irá representar o conteúdo interno que será vísivel ou não dependendo de sua seleção.

Todos os efeitos já serão gerados automaticamente se organizados os títulos e conteúdo dessa forma.

Vejamos um exemplo a seguir. Resultado no navegador:

Codigo:



```
<html>
  <head>
    <title></title>
    <meta charset="UTF-8">
    <script type="text/javascript" src="js/jquery.js"></script>
    <script type="text/javascript" src="js/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/jquery-ui.js"></script>
    <link rel="stylesheet" type="text/css" href="css/jquery-ui.min.css">
    <script type="text/javascript">
      $(document).ready(
        function () {
          $("#conteudo").accordion();
        });
    </script>
  </head>
  <body>
    <div id="conteudo" style="width=500px">
      <h3> Artigo I</h3>
      <div>
        <p>
          Blockchain talvez seja o recurso mais interessante da criptomoeda, com sua capacidade de registrar transações
          com segurança e publicamente. Considerando que o mercado de ações é uma confusão de troca de negociações, usar
          o blockchain para essas transações pode ser uma ideia surpreendentemente boa.
        </p>
      </div>
      <h3> Artigo II</h3>
      <div>
        <p>
          O Overstock.com é uma loja online que abraçou o Bitcoin e o blockchain, e agora está levando o amor à
          criptografia a um novo nível: com aprovação da Comissão de Valores Mobiliários (SEC, na sigla em inglês) dos
          EUA, o Overstock planeja emitir títulos púbicos. É um grande passo para o blockchain como tecnologia,
          considerando que até hoje se manteve limitado ao Bitcoin.
        </p>
      </div>
      <h3> Artigo III</h3>
      <div>
        <p>
          O blockchain é uma plataforma para tornar registros de transações publicamente disponíveis e que não pode ser
          modificado. Usuários podem escrever transações na cadeia (e elas são verificadas por uma vasta rede de
          computadores), mas não podem, posteriormente, eliminar essas entradas. Na ausência de um banco central, é isso
          o que mantém o Bitcoin funcionando.
        </p>
      </div>
    </div>
  </body>
</html>
```

Além de texto, podemos colocar outros elementos HTML, como Imagens, links, tabelas, listas etc.

Podemos usar esses recursos para uma área de notícias ou artigos por exemplo, pesquise em outros sites novos contextos onde podemos utilizar **accordion** para facilitar o acesso à conteúdos WEB.

13.4 Tabs

Essa função é uma outra forma elegante de organizar o conteúdo de uma página WEB através de abas por link para visualização de conteúdo de forma dinâmicas. Vejamos sua sintaxe e como aplica-las.

$(“elemento”).**tabs();**

Onde esse elemento descrito na sintaxe poderá ser uma ID de uma camada. Para organizar o conteúdo interno, devemos estruturar da seguinte forma:

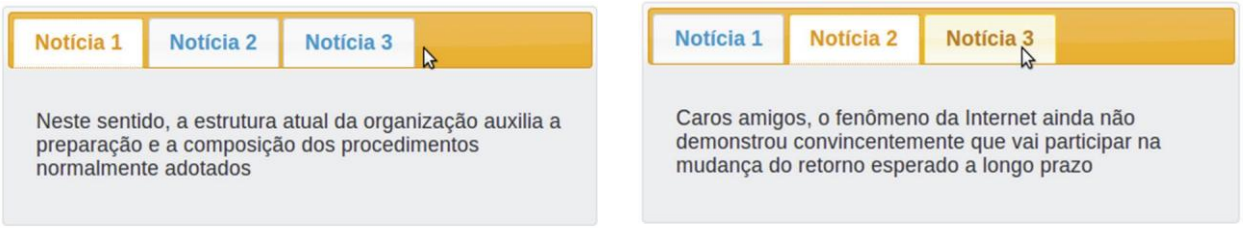
**
<a>:** Inicialmente criamos uma lista de links para representar as abas de conteúdo. Os índices deverão linkar um ID será responsável por alterar o conteúdo.

** :** Cada camada deverá ter uma indicação **“ID”** que irá representar um conteúdo.

Todos os efeitos já serão gerados automaticamente se organizados os títulos e conteúdo dessa forma: Vejamos um exemplo a seguir:

```
        <script>
           $(document).ready(
           function(){
             $("#noticias").tabs();

           });
        </script>
    </head>
    <body>
        <div id="noticias" style="width: 500px">
    <ul>
        <li><a href="#not-1">Notícia 1</a></li>
        <li><a href="#not-2">Notícia 2</a></li>
        <li><a href="#not-3">Notícia 3</a></li>
    </ul>
    <div id="not-1">
        <p>
            Neste sentido, a estrutura atual da organização auxilia a
            preparação e a composição dos procedimentos normalmente adotados
        </p>
    </div>
    <div id="not-2">
        <p>Caros amigos, o fenômeno da Internet ainda não demonstrou
            convincentemente que vai participar na mudança do retorno
            esperado a longo prazo</p>
    </div>
    <div id="not-3">
        <p>  Assim mesmo, a determinação clara de objetivos agrega valor ao
            estabelecimento de todos os recursos funcionais envolvidos</p>
    </div>
</div>
    </body>
```

Temos então o resultado a seguir no navegador:



13.5 Menu

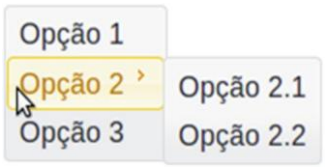
Esta função é responsável por montar um Menu DropDown de forma mais simples e elegante usando apenas listas e links como referência. Vejamos usa sintaxe:

```
$("elemento").menu();
```

Onde esse elemento descrito na sintaxe poderá ser uma ID de uma camada. Para organizar o conteúdo interno, devemos estruturar da seguinte forma:

<a>: monta um novo botão para link no menu. Para criar um menu DropDown bastar colocar uma lista dentro de outra lista, como fizemos no link "Opção 2" no link abaixo, veja o exemplo:

```
    <script>
        $(document).ready(
        function(){
            $("#menu").menu();
        });
    </script>
</head>
<body>
    <ul id="menu" style="width: 100px">
        <li>
            <a href="">Opção 1</a>
        </li>
        <li>
            <a href="">Opção 2</a>
            <ul>
                <li>
                    <a href="">Opção 2.1</a>
                </li>
                <li>
                    <a href="">Opção 2.2</a>
                </li>
            </ul>
        </li>
        <li>
            <a href="">Opção 3</a>
        </li>
    </ul>
</body>
```



Temos então o resultado a seguir no navegador:

13.6 Tooltip

Essa função é responsável por indicar aos usuários alguma informação relevante para o preenchimento de algum dado do formulário. Esse método já existe usando apenas o HTML, porém o jQuery pode tornar esse recurso mais elegante. Vejamos sua sintaxe:

```
$("elemento").tooltip();
```

Você poderá utilizar "document" como parametro do "elemento". Assim todos os campos de um formulário que tiver "title" será alterado com o novo designer baseado no tema escolhido do jQuery.

```
        <script>
           $(document).ready(
           function(){
               $(document).tooltip();

           });
        </script>
    </head>
    <body>
        <form name="formulario" action="#" method="POST">
            <p>Nome:
                <input id="nome" size="20"/></p>
            <p>CPF :
            <input id="cpf" title="Digite seu CPF sem pontos e hifen"/></p>
            <input type="submit" value="enviar" name="enviar" />
            <input type="reset" value="limpar" name="limpar" />
        </form>
    </body>
```

Temos então o resultado a seguir no navegador: