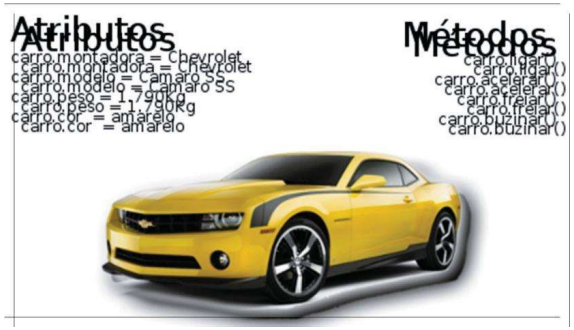
## 6.0 Objetos em JavaScript

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

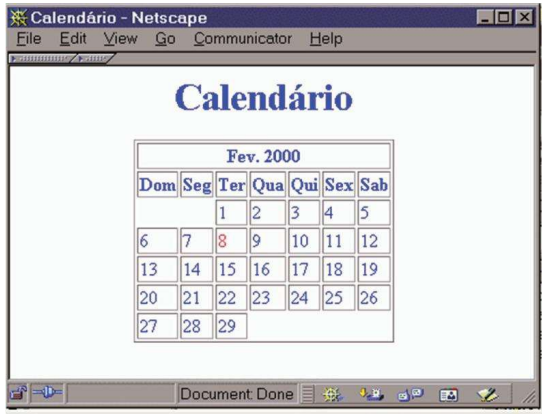
Netscape. Agora se utiliza setFullYear(), indicando o ano com todos os dígitos.

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.