

# Curso de Javascript

# JS



Este material é exclusivo do canal (youtube): [canalfessorbruno](https://www.youtube.com/c/canalfessorbruno) e do site [www.cfbcursos.com.br](http://www.cfbcursos.com.br)  
não pode ser distribuído ou comercializado por outra fonte.

**Apostila: versão 1.0**  
**29/01/2016**

**[www.youtube.com/canalfessorbruno](http://www.youtube.com/canalfessorbruno)**

**[www.cfbcursos.com.br](http://www.cfbcursos.com.br)**

**[canalfessorbruno@gmail.com](mailto:canalfessorbruno@gmail.com)**

**[www.facebook.com/canalfessorbruno](http://www.facebook.com/canalfessorbruno)**

**twitter: @fessorBruno**

## Sumário

Introdução.....	7
Comando de escrita – document.write .....	7
Onde inserir os scripts?.....	10
Comentários.....	10
Variáveis e constantes .....	11
Var .....	11
Const .....	15
console.log().....	16
Caixas de mensagens .....	16
Alert.....	17
Prompt .....	17
Confirm .....	19
Estrutura de decisão IF.....	20
IF.....	20
IF – ELSE .....	22
&& (and = E) -    (or = ou) .....	23
IF aninhado.....	25
Blocos de código, uso de { } .....	26
Switch - case.....	28
Array.....	30
Unidimensional / Vetor.....	30
Bidimensional / Matriz.....	33
Incremento e decremento de variáveis.....	36
Pós e Pré incremento .....	38
Loops / Estruturas de repetição.....	39
For / Para.....	39
While / Enquanto .....	43
Do While / Faça Enquanto .....	44
Loops infinitos – cuidado .....	45
Try – Catch – Finally .....	46
getElementById.....	47
innerHTML .....	48
getElementsByTagName .....	49
querySelectorAll.....	50
Acessando elementos de formulários .....	53

Date - Trabalhando com data e hora .....	53
Math.....	55
Function .....	57
Escopo, variáveis locais e globais.....	63
Eventos.....	66
Eventos de mouse .....	67
Eventos de teclado.....	67
Eventos de objetos/frames/body .....	67
Eventos de formulários .....	68
Eventos de Drag / Arrastar.....	68
Eventos de Área de transferência / clipboard .....	68
Eventos de impressão .....	68
Eventos de mídia.....	68
Eventos de animação .....	69
Evento de transição .....	69
Eventos enviados pelo servidor .....	69
Eventos diversos .....	69
Eventos de toque na tela .....	70
Event .....	70
MouseEvent .....	70
KeyboardEvent.....	70
HashChangeEvent .....	71
PageTransitionEvent .....	71
FocusEvent .....	71
AnimaionEvent.....	71
TransitionEvent .....	71
WheelEvent.....	71
addEventListener .....	73
removeEventListener().....	77
Eventos de controle de tempo/intervalo (cronometragem) / timing events.....	77
setTimeout .....	78
setInterval .....	78
clearInterval .....	79
setInterval como enterFrame .....	81
Criando um relógio digital.....	83
Recursividade.....	85

Validação de formulários .....	89
Javascript x CSS .....	93
Formatação condicional.....	102
Função para cálculo de IMC.....	105
replace() .....	106
toFixed() .....	107
Slider simples .....	107
Adicionando e ou removendo atributos HTML via javascript.....	110
setAttribute() .....	110
removeAttribute().....	111
hasAttribute().....	112
String – Funções para manipular strings.....	112
match() .....	113
search() .....	118
replace() .....	118
charAt() .....	119
charCodeAt() .....	119
concat() .....	120
fromCharCode() .....	120
indexOf() .....	121
lastIndexOf().....	121
localeCompare().....	121
slice() .....	121
split() .....	122
substr() .....	122
toLowerCase() .....	123
toUpperCase() .....	123
toString() .....	123
trim() .....	124
Usando caracteres especiais na string .....	124
Jogo Ping-Pong For One .....	125
Objeto window.....	129
Objeto navigation.....	131
Objeto screen.....	132
Objeto history .....	133
Objeto location .....	133

Adicionando, removendo e modificando elementos na página .....	134
createElement - método .....	134
createTextNode - método .....	136
createAttribute - método .....	137
insertBefore - método .....	138
replaceChild - método .....	139
childNodes – propriedade .....	140
parentNode – propriedade .....	141
remove() – método .....	142
getAttribute() – método .....	145
activeElement - propriedade .....	146
hasFocus - método .....	146
Datasets .....	147
scrollIntoView .....	148
Atributo hidden .....	148
Código javascript externo – Arquivos .js .....	149
Cookies .....	149
Considerações finais .....	150

## Introdução

A primeira observação a ser feita no curso de Javascript é que Java e Javascript não são a mesma coisa, não tem nenhuma relação entre as duas linguagens.

Javascript é classificada como uma linguagem multiplataforma, isso significa que Javascript não é exclusiva da web, um bom exemplo disso que é Javascript também é usada como linguagem em Unity3D.

Neste curso vamos aprender Javascript para web, criando scripts para incrementar as páginas web junto ao código HTML.

## Comando de escrita – document.write

Como nosso curso será voltado para web, precisamos de uma página HTML para trabalhar com Javascript, inicialmente iremos trabalhar com código incorporado na própria página.

Vamos usar o “Notepad++” como editor padrão, por possuir algumas facilidades, mas o bloco de notas pode ser usado sem problema nenhum.

Crie um arquivo com o código básico HTML mostrado a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      //Conteúdo Javascript
    </script>
  </head>
  <body>
    <!-- Conteúdo da página -->
  </body>
</html>
```

Uma linguagem de programação é fundamental ter um comando para saída de texto, no caso de Javascript temos o método “write”, vamos entender esse comando.

Veja a sintaxe básica do comando.

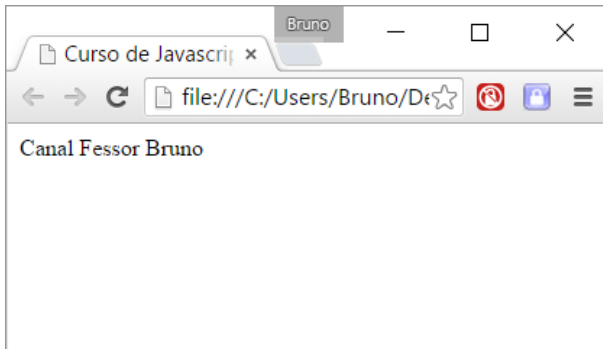
```
document.write("Texto a ser mostrado");
```

Dentro dos parênteses inserimos o conteúdo que será impresso no browser, quando o conteúdo não for um cálculo ou o valor de uma variável, quando for um texto simples, deve vir entre aspas, basicamente é o que chamamos de string.

Adicione o código destacado em vermelho no código a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      document.write("Canal Fessor Bruno");
    </script>
  </head>
  <body>
    <!-- Conteúdo da página -->
  </body>
</html>
```

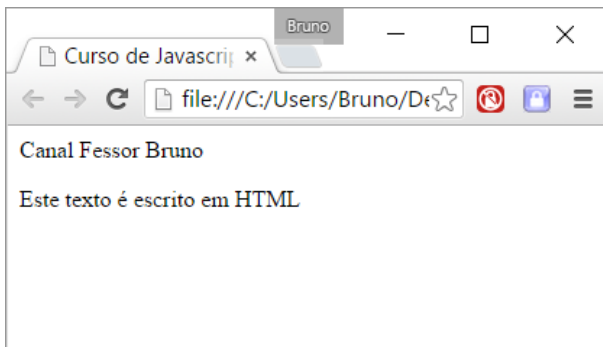
Salve as alterações e abra o arquivo em seu browser. O resultado será o texto “Canal Fessor Bruno” sem aspas escrito em seu browser.



Vamos adicionar um parágrafo como o código destacado em vermelho a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      document.write("Canal Fessor Bruno");
    </script>
  </head>
  <body>
    <p>Este texto é escrito em HTML</p>
  </body>
</html>
```

Ao salvar as alterações e atualizar a página teremos o resultado conforme a ilustração a seguir.



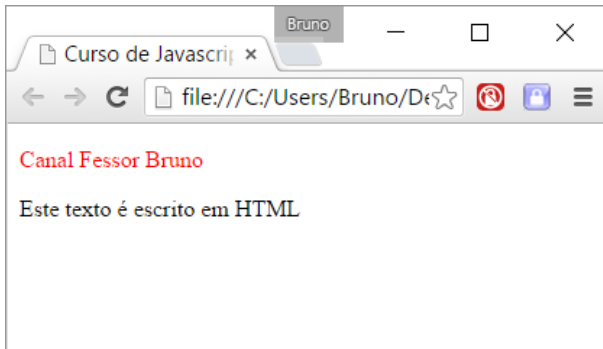
A diferença é que o texto “Canal Fessor Bruno” é inserido via Javascript e o texto “Este texto é escrito em HTML” em parágrafo padrão HTML.

O comando write também pode inserir código HTML interpretável pelo browser, basta inserir normalmente as tags entre as aspas, veja o código em vermelho a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      document.write("<p style='color:#F00'>Canal Fessor Bruno</p>");
    </script>
  </head>
  <body>
    <p>Este texto é escrito em HTML</p>
  </body>
</html>
```



O comando anterior o método write insere o texto em formatação de parágrafo <p> na cor vermelho, veja a ilustração a seguir.

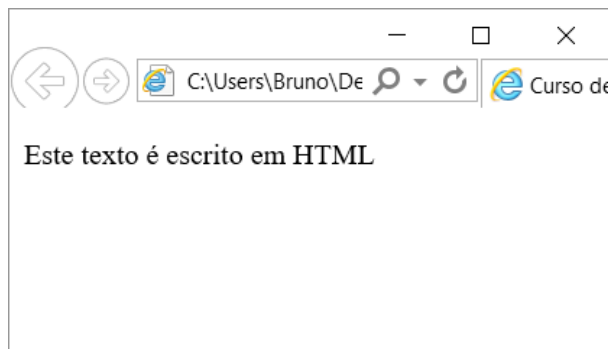
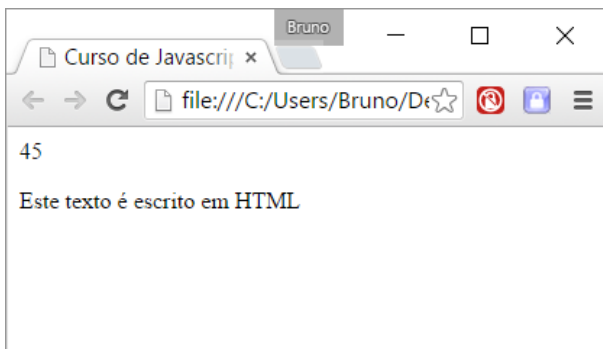


Outra observação importante é a diferenciação de textos e numerais, por exemplo, 50 e "50" são diferentes, quando usamos aspas, como já foi dito o conteúdo é tratado como texto/string, quando está sem aspas é um numeral e pode ser calculado. Veja o código a seguir

```
document.write("50"-5);
```

De uma forma geral este código está errado e o browser não consegue realizar este cálculo, porém, o Google Chrome, por exemplo, "pula" o erro, na verdade ele verifica que mesmo sendo uma string o conteúdo é um numeral e realiza o cálculo sem gerar erro, já o Internet Explorer não realiza o cálculo e não executa o comando, lembre-se que esse é o comportamento padrão.

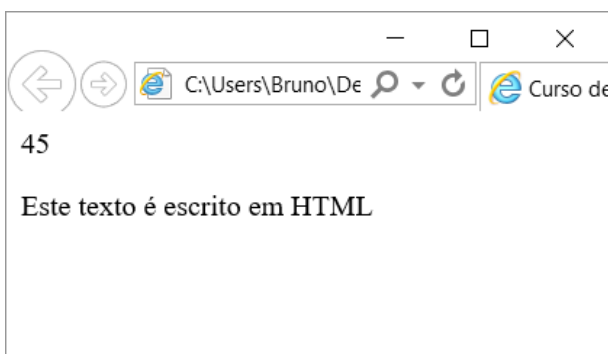
Veja as ilustrações a seguir do Google Chrome à esquerda e Internet Explorer à direita, note que o Chrome mostra o cálculo e o I.E. não mostra.



Portanto a forma correta para este cálculo é o código a seguir.

```
document.write(50-5);
```

Neste caso o I.E. realiza o cálculo como mostrado a seguir.



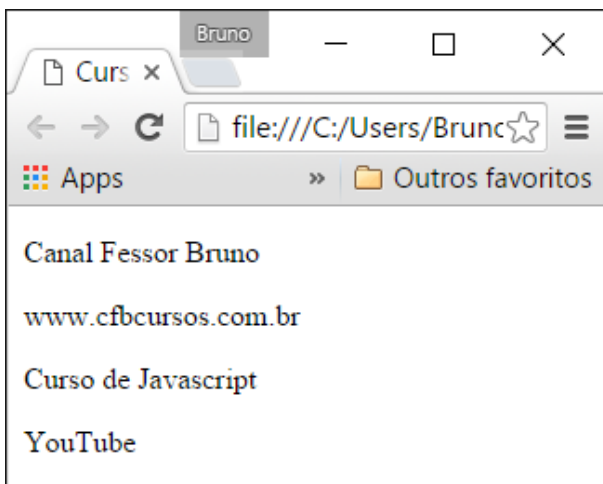
## Onde inserir os scripts?

Os scripts podem ser inseridos dentro do <head> da página como uma forma mais global ou em qualquer momento dentro da tag <body>, desde que sempre estejam dentro da tag <script>, veja o código de exemplo a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      document.write("<p>Canal Fessor Bruno</p>");
    </script>
  </head>
  <body>
    <script>
      document.write("<p>www.cfbcursos.com.br</p>");
    </script>

    <p>Curso de Javascript</p>

    <script>
      document.write("<p>YouTube</p>");
    </script>
  </body>
</html>
```



## Comentários

Comentários são blocos de códigos que não serão interpretados ou compilados, em javascript podemos usar duas formas de inserir ou comentar códigos, veja a seguir.

//Comentário de linha única

/\*

Comentário

De várias linhas

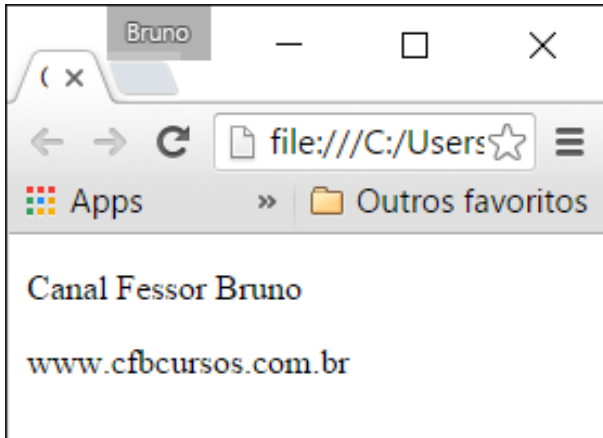
\*/

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      document.write("Canal Fessor Bruno");
      //document.write("Curso de Javascript");
      document.write("www.cfbcursos.com.br");
    </script>
  </head>
  <body>
    <p>Curso de Javascript</p>
  </body>
</html>
```

```
        /*
        document.write("CFB");
        document.write("YouTube");
        */
    </script>
</head>
<body>

</body>
</html>
```

Veja o resultado do código acima.



Note que foram impressas somente duas linhas de texto, isso porque as demais estão em forma de comentário. Todo o conteúdo que estiver em comentário é simplesmente ignorado e não é executado.

Você pode usar comentário em qualquer local do código javascript e este inclusive é um ótimo hábito, pois irá facilitar na identificação das partes do seu código.

## Variáveis e constantes

É muito importante em uma linguagem de programação poder armazenar dados de forma temporária, essa é a função das variáveis, sempre que precisar armazenar uma informação, vamos usar as variáveis, podemos inserir, remover ou alterar informações em variáveis, vamos ver como usar esse recurso fundamental.

### Var

Declarar variáveis em javascript é bem simples, diferente de linguagens tradicionais como C++, em javascript não informamos o tipo de dados e por isso não precisamos de operações como typecast, basta usar a palavra reservada “var” e o nome da variável, veja a seguir o código básico para se declarar uma variável.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var jogador;
      var vidas;
      var municao;
    </script>
  </head>
  <body>

  </body>
</html>
```

No código acima vimos a forma tradicional de declarar variáveis, muitas pessoas usam esta maneira simplesmente por facilitar a leitura e identificação, mas podemos declarar todas em uma só linha, para economizar código, como podemos ver no exemplo a seguir.

```
<script>
  var jogador,vidas,municao;
</script>
```

Um detalhe importante que já ressaltai é que não precisamos informar o tipo de dados que a variável irá receber e nem precisamos de typecast, portanto, podemos inserir um tipo de dados “string” por exemplo e logo a seguir podemos inserir um numeral “int” sem nenhum problema.

Vamos ver como atribuímos valores às variáveis.

```
<script>
  var jogador,vidas,municao;

  jogador="Bruno";
  vidas=3;
  municao=100;
</script>
```

Veja que atribuição é simples, usamos o operador = (atribuição) e como já foi destacado anteriormente, se o valor inserido for um texto deve vir entre aspas “texto”.

Ressaltando o detalhe do typecast automático o exemplo a seguir é perfeitamente válido.

```
jogador="Bruno";
jogador=0.6;
jogador=100;
```

Um detalhe a observar é que embora usamos três valores diferentes, string, float e int, o valor que a variável armazenará é sempre o último que foi atribuído, neste caso o valor da variável será 100, um valor vai substituindo o outro.

Podemos atribuir valores imediatamente no momento da declaração, ao declarar uma variável inicialmente ela tem valor indefinido “undefined”.

Veja o código a seguir que mostra como atribuir valor à variável imediatamente após a declaração.

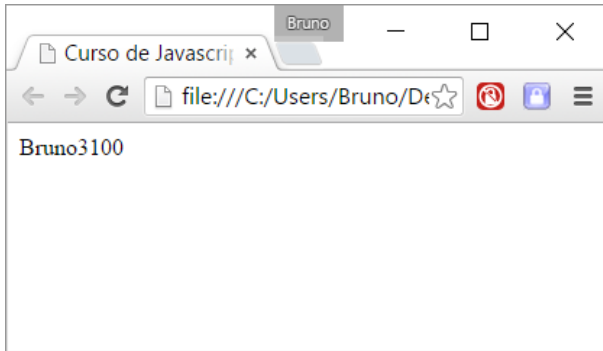
```
<script>
  var jogador="Bruno";
  var vidas=3;
  var municao=100;
</script>
```

Precisamos ver os valores das variáveis, então, temos que aprender como imprimir na tela o valor atual de uma variável, é simples e usamos o método write que aprendemos anteriormente, basta informar o nome da variável dentro dos parênteses como no exemplo a seguir.

```
<script>
  var jogador="Bruno";
  var vidas=3;
  var municao=100;

  document.write(jogador);
  document.write(vidas);
  document.write(municao);
</script>
```

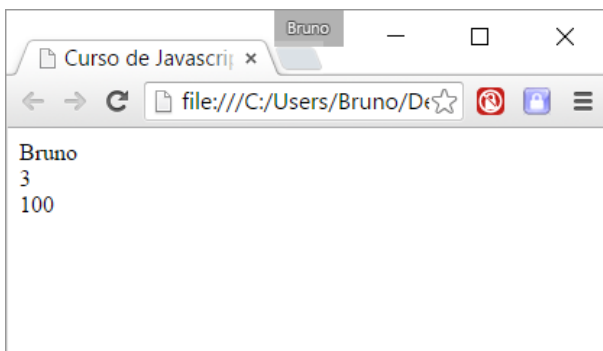
Veja o resultado do código acima.



Naturalmente os valores foram impressos um após o outro, sem espaços ou quebras de linha, esse é o comportamento padrão, se precisarmos de espaços e ou quebras de linhas precisamos inserir como código HTML, como no código a seguir.

```
document.write(jogador + "<br>");  
document.write(vidas + "<br>");  
document.write(municao + "<br>");
```

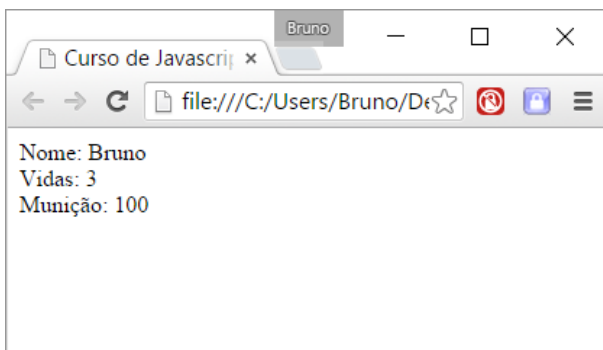
Nos comandos acima inserimos o conteúdo da variável mais o código HTML para quebra de linha, neste caso o operador + “soma” funciona como concatenação, o resultado deste código pode ser comparado com a ilustração a seguir.



Vamos adicionar um pouco mais de detalhe em nossa impressão.

```
document.write("Nome: " + jogador + "<br>");  
document.write("Vidas: " + vidas + "<br>");  
document.write("Munição: " + municao + "<br>");
```

Nos comandos write acima usamos duas strings e uma variável para impressão (string + variável + string HTML).



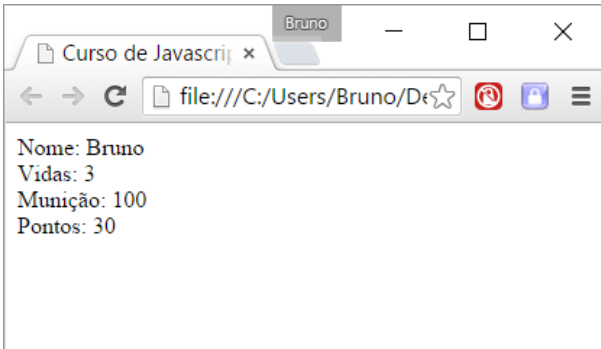
Vamos aprender mais uma possibilidade de impressão, misturando strings com cálculos, dê uma olhada no código em vermelho seguir e note que iniciamos a variável pontos com valor zero e no momento da impressão realizamos um cálculo que resultará no valor 30.

```
<script>  
var jogador="Bruno";
```

```
var vidas=3;
var municao=100;
var pontos=0;

document.write("Nome: " + jogador + "<br>");
document.write("Vidas: " + vidas + "<br>");
document.write("Munição: " + municao + "<br>");
document.write("Pontos: " + (pontos + (vidas * 10)) + "<br>");
</script>
```

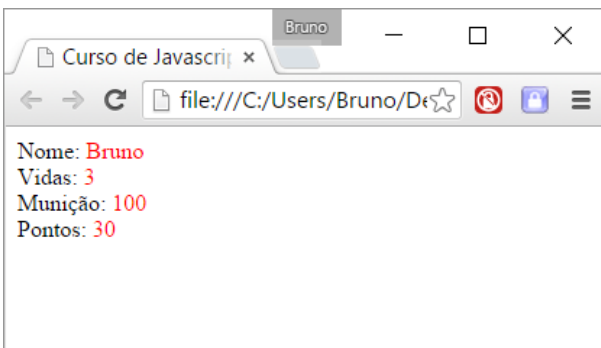
Confira o resultado na ilustração a seguir.



Vamos incrementar mais um pouco o código, vamos adicionar formatação CSS inline, diretamente no HTML para formatar o valor das variáveis em vermelho.

```
document.write("Nome: <span style='color:#F00'>" + jogador + "</span><br>");
document.write("Vidas: <span style='color:#F00'>" + vidas + "</span><br>");
document.write("Munição: <span style='color:#F00'>" + municao + "</span><br>");
document.write("Pontos: <span style='color:#F00'>" + (pontos + (vidas * 10)) + "</span><br>");
```

Veja o resultado do código acima na ilustração a seguir.

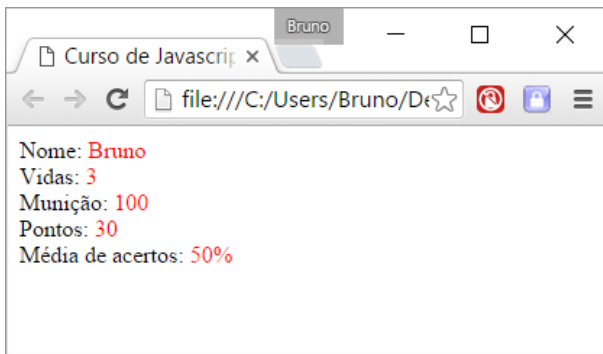


Os cálculos também podem ser realizados na atribuição das variáveis, veja o código seguir destacado em vermelho.

```
<script>
var jogador="Bruno";
var vidas=3;
var municao=100;
var pontos=0;
var tiros=1500;
var acertos=750;
var mediaDeAcertos=(acertos/tiros)*100;

document.write("Nome: <span style='color:#F00'>" + jogador + "</span><br>");
document.write("Vidas: <span style='color:#F00'>" + vidas + "</span><br>");
document.write("Munição: <span style='color:#F00'>" + municao + "</span><br>");
document.write("Pontos: <span style='color:#F00'>" + (pontos + (vidas * 10)) + "</span><br>");
document.write("Média de acertos: <span style='color:#F00'>" + mediaDeAcertos + "</span><br>");
</script>
```

O código acima declara uma variável e atribui o resultado do cálculo da média de disparos que acertaram o alvo, e é claro imprime este valor.



## Const

Constantes tem uma diferença importante em relação às variáveis, elas não podem ser modificadas, uma vez declarada a atribuída uma constantes, seu valor não pode mais ser modificado.

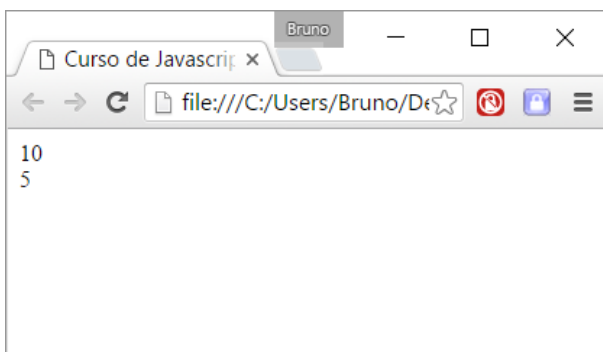
A sintaxe de declaração é a mesma da variável com exceção da palavra “var” que será substituída pela palavra “const”, veja o código a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      const max=10;
      const min=5;

      document.write(max + "<br>" + min);
    </script>
  </head>
  <body>

  </body>
</html>
```

O resultado desse código é simplesmente a impressão do valor das constantes na tela.



Uma constante pode ser usada em qualquer lugar do programa, como uma variável, podemos utilizar em cálculos e impressões, a única coisa que não podemos fazer é alterar seu valor. Portanto o código a seguir destacado em vermelho não terá funcionalidade alguma e está incorreto.

```
<script>
  const max=10;
  const min=5;

  max=8;

  document.write(max + "<br>" + min);
</script>
```

## console.log()

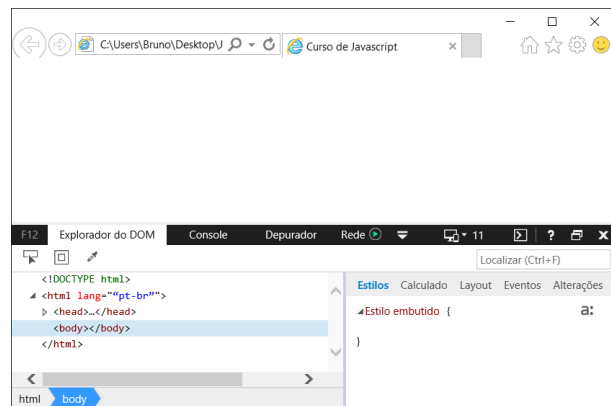
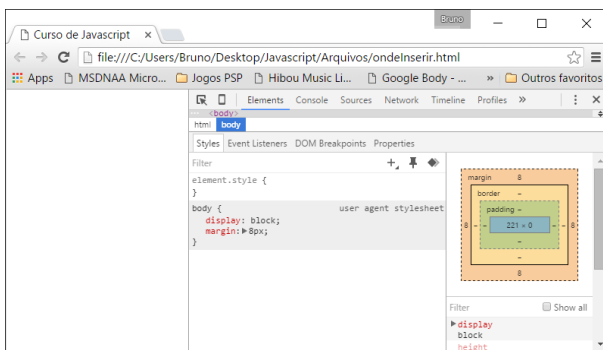
Podemos ativar o modo console para testar variáveis e outros dados em nosso código, por exemplo, se quiser saber o valor de uma variável sem ter que imprimir na tela.

Para ativar o modo console geralmente usamos a tecla “F12”, veja o código a seguir.

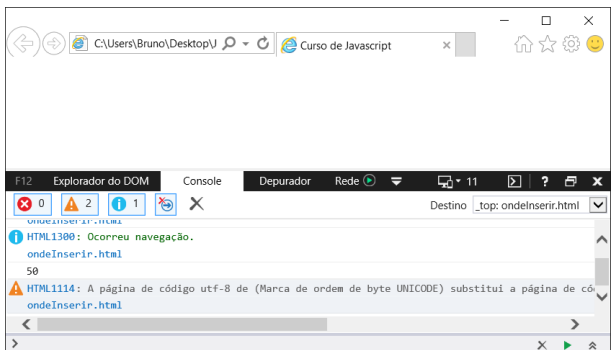
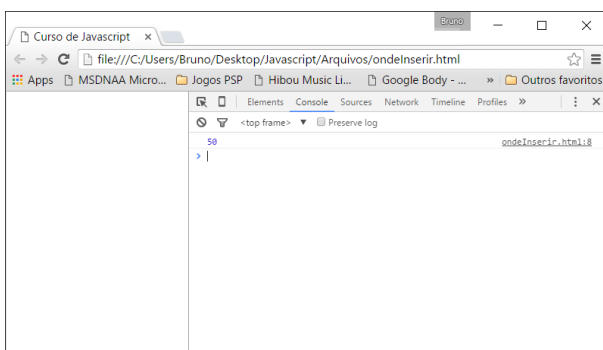
```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      num=20+30
      console.log(num);
    </script>
  </head>
  <body>

  </body>
</html>
```

Na tela não é impresso nada, mas vamos ativar o modo console, pressione a tecla “F12” e será aberto o modo console, a seguir é mostrado no Chrome à esquerda e no I.E. à direita.



Em ambos vamos encontrar a guia “Console”, clique nesta aba e veja que estará mostrando o valor da variável “num”.



## Caixas de mensagens

Uma ótima forma de comunicação com o usuário ou uma forma alternativa de colher dados para nosso programa são as caixas de mensagens, em javascript existem três tipos diferentes, são elas alert, prompt e confirm, vamos entender as três neste capítulo.



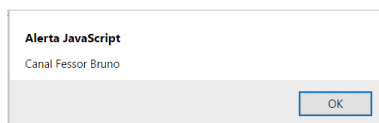
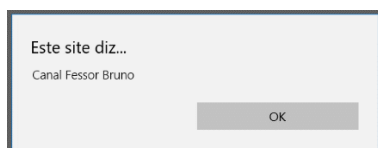
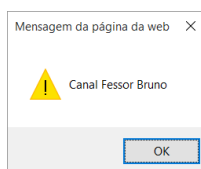
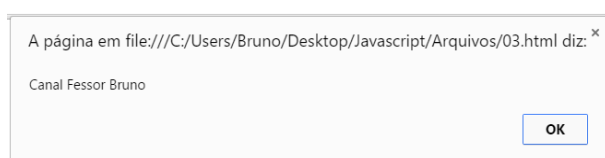
## Alert

A caixa de mensagem mais simples é a caixa “alert”, mostra uma mensagem simples com um botão somente o botão OK, o comando é bem simples, basta inserir o conteúdo que será mostrado na caixa de texto dentro dos parênteses, veja o código a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      alert("Canal Fessor Bruno");
    </script>
  </head>
  <body>

</body>
</html>
```

Veja a seguir o resultado do código acima em quatro navegadores diferentes, Google Chrome, Internet Explorer, Microsoft Edge, Opera.



Como vimos a caixa alert é meramente informativa, um detalhe interessante é que podemos usar a mesma lógica usada no método write para configuração do texto a ser mostrado, vamos incrementar nosso código com uma variável e mostra-la na caixa de mensagem alert.

Veja o código a seguir.

```
<script>
  var canal="youtube.com/canalfessorbruno";
  alert("Canal Fessor Bruno\n" + canal);
</script>
```

Veja que no código do método alert inserimos uma string e o conteúdo da variável canal, mas ainda tem um detalhe importante que devemos observar, veja que no final da string existe um \n que é o comando para quebra de linha, sempre que encontrar o comando \n será quebrada uma linha, então veja o resultado na ilustração a seguir.



## Prompt

A caixa de mensagens prompt se diferencia por possibilitar a entrada de texto, permitindo que esse texto seja coletado e passado a uma variável, deste maneira é a primeira forma de coletar dados externos que iremos aprender.

A sintaxe de utilização do prompt é a seguinte.

```
prompt("Texto principal a ser mostrado","Valor inicial mostrado na caixa de texto");
```

Visualmente a caixa prompt se destaca por ter um campo para entrada de texto, o valor digitado neste campo será o valor de retorno do método, podendo ser armazenado então em uma variável, veja o código a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var nome;
      nome=prompt("Por favor digite seu nome","Digite seu nome aqui");
      document.write(nome);
    </script>
  </head>
  <body>

  </body>
</html>
```

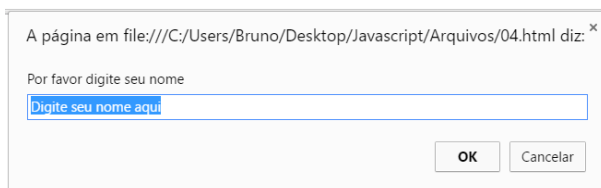
Note que inicialmente criamos uma variável chamada “nome” como não inserimos nenhum valor a ela, por padrão, está definida com valor “undefined”.

Em seguida usamos o método prompt armazenando o valor digitado na variável nome.

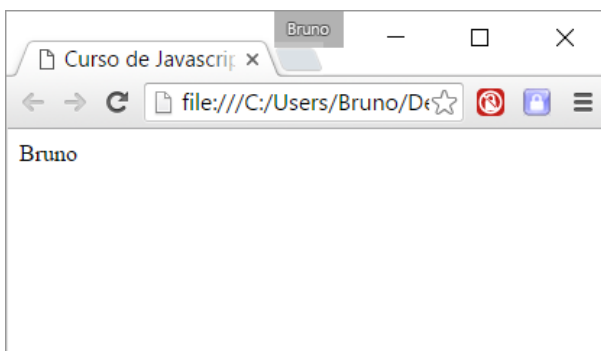
```
nome=prompt("Por favor digite seu nome","Digite seu nome aqui");
```

Na última linha do script simplesmente escrevemos na tela o valor da variável nome.

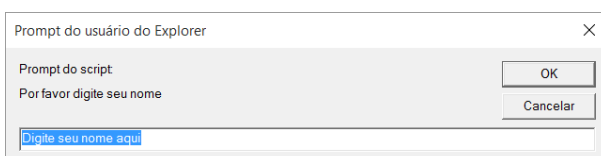
Veja a seguir a ilustração mostrando o resultado deste código.



Ao digitar o nome e clicar no botão OK, será impresso na tela o nome digitado.



Veja a ilustração a seguir mostrando como é a caixa prompt renderizada pelo Internet Explorer e pelo Opera.



**JavaScript**  
Por favor digite seu nome  
  

OK Cancelar

## Confirm

A caixa de mensagens confirme se particulariza por apresentar uma mensagem simples, dois botões OK e Cancelar.

Mas sua funcionalidade mais importante é o retorno de qual dos botões foi pressionado, caso seja clicado em OK a caixa retorna o valor “true” e caso seja clicado Cancelar a caixa retorna o valor “false”, simples assim.

Para tomar uma decisão após o clique basta usarmos o comando IF que veremos logo adiante, mas podemos armazenar este retorno também em uma variável como mostra o código a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var opc;
      opc=confirm("Deseja continuar?");
      document.write(opc);
    </script>
  </head>
  <body>

  </body>
</html>
```

Criamos uma variável chamada “opc” sem valor definido.

Usamos a variável opc para receber o retorno do método confirm, lembre-se, se o botão OK for clicado o método retorna “true” (verdadeiro) e se Cancelar for clicado o método retorn “false” (falso), tornando assim a variável opc como do tipo boolean.

Na sequência imprimimos o valor da variável opc na tela, faça o teste clicando nos dois botões, veja o resultado do código a seguir.

A página em file:///C:/Users/Bruno/Desktop/Javascript/Arquivos/05.html diz: ✖  
Deseja continuar?  

OK Cancelar

Clique no botão OK

Curso de Javascript x Bruno

file:///C:/Users/Bruno/D...

true

Clique no botão Cancelar

Curso de Javascript x Bruno

file:///C:/Users/Bruno/D...

false

## Estrutura de decisão IF

Em qualquer linguagem de programação é extremamente importante podermos analisar um certo conteúdo ou resultado para direcionar a execução do nosso programa e determinar uma rotina ou outra a ser executada.

### IF

A principal estrutura de decisão em programação é definida pelo comando IF, que traduzindo do inglês é SE, sua sintaxe básica é bem simples e vou mostrar a seguir.

```
if(teste lógico){
    //comandos a serem executados se o teste for verdadeiro
}
```

Nesta sintaxe o IF verifica o resultado do “teste lógico” e se for verdadeiro “true” ele executa os comandos entre suas chaves { }, caso contrário, se for falso, os comandos não serão executados e pulará para a próxima instrução após o IF.

O que é o “teste lógico”?

Basicamente o comando IF necessita de uma expressão que retorne “verdadeiro (true)” ou “falso (false)” e estes são os valores de retorno de qualquer teste lógico, basicamente é uma comparação que usa os operadores lógicos descritos a seguir:

Operador	Nome	Exemplo
>	Maior	10 > 5
<	Menor	5 < 10
>=	Maior ou igual	10 >= 10
<=	Menor ou igual	10 <= 10
!=	Diferente	1 != 2
==	Igualdade	1 == 1

Vou mostrar alguns exemplos de testes lógicos.

Teste	Retorno true=verdadeiro=1 / false=falso=0
10 > 5	True
5 > 10	False
10 < 5	False
5 < 10	True
10 >= 10	True
10 >= 5	True
5 >= 10	False
10 <= 10	True
10 <= 5	False
5 <= 10	True
10 != 10	False
10 != 5	True
10 == 10	True
10 == 5	False
0	True
1	False

Podemos usar os numerais 0 e 1 diretamente em testes lógicos, o numeral 0 (zero) para false e o numeral 1 (um) para true.

Vamos a um exemplo.

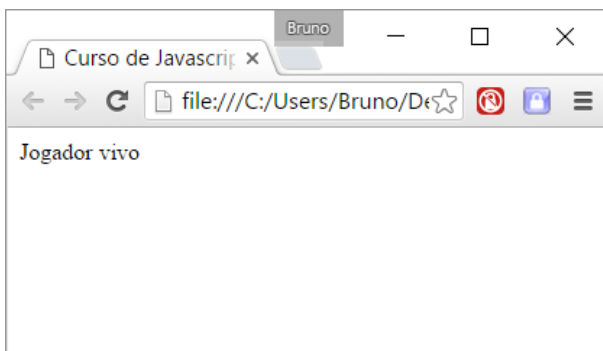
```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var vidas=3;
      if(vidas > 0){
        document.write("Jogador vivo");
      }
    </script>
  </head>
  <body>

  </body>
</html>
```

O código acima cria uma variável chamada vidas e a inicia com o valor 3.

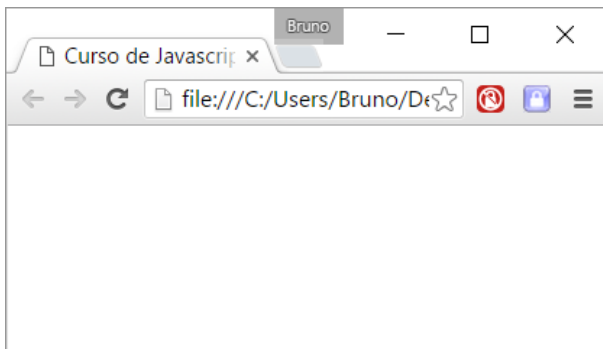
Em seguida o IF compara o valor da variável vidas, assim, verificando se o valor da variável é maior que zero, caso seja, o comando do bloco deste IF, dentro das chaves { }, será executado.

Neste caso o teste retorna verdadeiro, então, será impresso na tela o texto “Jogador vivo”, veja o resultado na ilustração a seguir.



Mude o valor da variável para 0 como o código a seguir e veja o resultado.

```
<script>
  var vidas=0;
  if(vidas > 0){
    document.write("Jogador vivo");
  }
</script>
```



Note que agora não temos nenhum texto como mostra a ilustração acima, isso porque após o IF não existe comando nenhum e como o teste lógico do IF retornou falso, pois, o valor de vidas não é maior que zero, o comando write dentro do IF não foi executado.

## IF – ELSE

Podemos incrementar nosso comando IF com o uso do ELSE, que pode ser entendido como “caso contrário / se não”.

Veja bem, falamos que os comandos do IF são executados somente quando o teste for verdadeiro correto? E se o valor do teste for falso? Ai é que entra o ELSE, os comandos do ELSE são executados sempre que o teste tiver um resultado falso, veja a sintaxe.

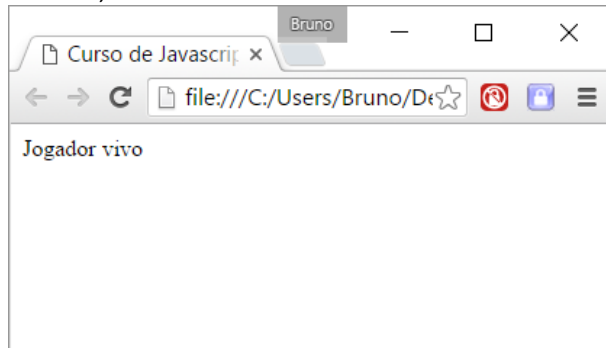
```
if(teste lógico){
    //comandos a serem executados se o teste for verdadeiro
}else{
    //comandos a serem executados se o teste for falso
}
```

Vamos alterar o código do IF para incrementar o ELSE.

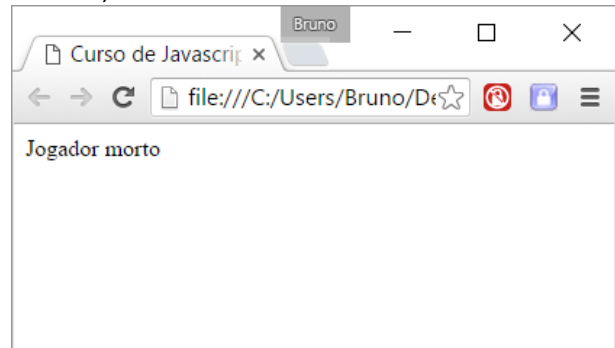
```
<script>
var vidas=3;
if(vidas > 0){
    document.write("Jogador vivo");
}else{
    document.write("Jogador morto");
}
</script>
```

Agora nosso programa vai imprimir um texto se o teste for verdadeiro ou falso, vamos ver os resultados.

vidas=3;



vidas=0;



Vamos incrementar um pouco mais nosso IF-ELSE, podemos adicionar um resultado para cada valor de vida, por exemplo, 3 = vivo, 2 = ferido, 1 = muito ferido, 0 = morto, vamos a esta implementação.

```
<script>
var vidas=3;
if(vidas == 3){
    document.write("Jogador vivo");
}else if(vidas == 2){
    document.write("Jogador ferido");
}else if(vidas == 1){
    document.write("Jogador muito ferido");
}else if(vidas == 0){
    document.write("Jogador morto");
}
</script>
```

Em programação existem várias maneiras de realizar uma mesma tarefa, tudo depende de sua lógica, vou mostrar outra maneira de implementar esse IF com outros operadores lógicos, confira o código a seguir.

```
<script>
  var vidas=3;
  if(vidas > 2){
    document.write("Jogador vivo");
  }else if(vidas == 2){
    document.write("Jogador ferido");
  }else if(vidas == 1){
    document.write("Jogador muito ferido");
  }else if(vidas < 1){
    document.write("Jogador morto");
  }
</script>
```

Reduzindo um pouco mais o código, veja o código a seguir, procure a mudança e pense porque o resultado é o mesmo.

```
<script>
  var vidas=3;
  if(vidas > 2){
    document.write("Jogador vivo");
  }else if(vidas == 2){
    document.write("Jogador ferido");
  }else if(vidas == 1){
    document.write("Jogador muito ferido");
  }else{
    document.write("Jogador morto");
  }
</script>
```

Percebeu a mudança? Está no último ELSE, note que não temos um teste IF, mas porquê? Simples, porque não precisa, caso todos os testes anteriores não sejam os verdadeiros, significa que o valor da vida não é 3, nem 2 e nem 1, então só pode ser zero, correto? Sim, então, não precisa ser testado, pois, é a única possibilidade.

### **&& (and = E) - || (or = ou)**

Ótimo até agora, mas vamos mudar um pouco, vamos muda de vidas para energia e ao invés de testar um único valor como 3, 2, 1 ou 0, vamos testar faixas de valores como de 1 a 20, de 21 a 40, de 41 a 60, de 61 a 80, de 81 a 100 e 0.

Neste caso vamos utilizar os operadores && ( E ) e || ( OU ) para incrementar nossas comparações, vamos ao código.

```
<script>
  var energia=100;
  var velocidade;
  if((energia > 80)&&(energia <= 100)){
    document.write("Energia alta");
    velocidade=100;
  }else if((energia > 60)&&(energia < 81)){
    document.write("Energia boa");
    velocidade=80;
  }else if((energia > 40)&&(energia < 61)){
    document.write("Jogador media");
    velocidade=60;
  }else if((energia > 20)&&(energia < 41)){
    document.write("Jogador baixa");
    velocidade=40;
  }else if((energia > 1)&&(energia < 21)){
    document.write("Jogador critica");
    velocidade=20;
  }else{
    document.write("Jogador morto");
    velocidade=0;
  }
</script>
```

Neste código acima, testamos o valor da variável energia e de acordo com o resultado imprimimos um texto na tela e alteramos o valor da variável velocidade.

Vou mostrar uma forma melhor de entender e ler este IF.

```

if((energia > 80)&&(energia <= 100)){
  se((energia for maior que 80) E (energia for menor ou igual a 100)){
    Se energia for maior que 80 e energia for menor ou igual a 100
    Se energia estiver entre 81 e 100

  else if((energia > 60)&&(energia < 81)){
    Caso contrário se((energia for maior que 60) E (energia for menor que 81)){
    Caso contrário se energia for maior que 60 e energia for menor que 81
    Caso contrário se energia estiver entre 61 e 80

  else if((energia > 40)&&(energia < 61)){
    caso contrário se((energia for maior que 40) E (energia for menor que 61)){
    caso contrário se energia for maior que 40 e energia for menor que 61
    Caso contrário se energia estiver entre 41 e 60
  
```

E assim por diante.

De uma forma bem resumida podemos ler estes comandos IF-ELSE da seguinte maneira.

Se energia estiver entre 81 e 100  
 Caso contrário se energia estiver entre 61 e 80  
 Caso contrário se energia estiver entre 61 e 80  
 Caso contrário se energia estiver entre 41 e 60  
 Caso contrário se energia estiver entre 21 e 40  
 Caso contrário se energia estiver entre 1 e 20  
 Caso contrário

Pare entendermos o || ( ou ) vamos criar outro código onde um jogador precise ter pontuação menor que 33 ou maior que 66 para vencer, qualquer pontuação entre 34 e 65, inclusive, o jogador irá perder o jogo.

```

<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var pontos=70;
      if((pontos < 34)|| (pontos > 65)){
        document.write("Jogador venceu");
      }else{
        document.write("Jogador perdeu");
      }
    </script>
  </head>
  <body>

  </body>
</html>
  
```

Vou detalhar este IF.

```

if((pontos < 34)|| (pontos > 65)){
  se((pontos for menor que 34) OU (pontos for maior que 65) ){
  se pontos for menor que 34 ou maior que 65
  
```

Neste IF os valores aceitos estão entre 0 a 33 e entre 66 e 100, na verdade qualquer valor acima de 65.

Vamos ver mais alguns exemplos de IF com AND e OR para fixar este recurso.

Vamos considerar as variáveis a seguir.



```
var n1=20;
var n2=40;
var n3=60;
var n4=80;
```

Vamos aos testes.

Teste	Retorno
if((n1 > 15)&&(n2 > 35)&&(n3 > 55)&&(n4 > 75))	True
if((n1 > 15)&&(n2 < 35)&&(n3 > 55)&&(n4 > 75))	False
if((n1 > 35)    (n2 > 55)    (n3 > 65)    (n4 > 85))	False
if((n1 > 35)    (n2 < 55)    (n3 > 65)    (n4 > 85))	True

Podemos descrever as tabelas verdade do AND e do OR para que fique mais claro (V=Verdadeiro / F=Falso).

&& - AND		
V	F	F
F	V	F
V	V	V
F	F	F

- OR		
V	F	V
F	V	V
V	V	V
F	F	F

Observando a tabela verdade do AND podemos ver que o único momento que AND retorna verdadeiro/true é quando todos os testes forem verdadeiros, se um dos testes retornar falso o retorno final será falso.

No OR podemos observar que o único momento que OR retorna falso/false é quando todos os testes forem falsos, se um dos testes retornar verdadeiro o retorno final será verdadeiro.

## IF aninhado

A prática de aninhar estruturas não é exclusiva do IF, podemos utilizar este recurso com diversos outros comandos, mas vamos utilizar o IF para entendermos o que é aninhar um comando.

Basicamente IF aninhado significa um IF dentro do outro, diferente de IF-ELSE, veja dois exemplos clássicos de IF aninhado a seguir.

### Exemplo 1 – IF aninhado

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var nota=95;
      if(nota < 40){
        document.write("Reprovado com louvor.");
      }else{
        if(nota < 60){
          document.write("Reprovado por nota baixa.");
        }else{
          if(nota < 80){
```

```
        document.write("Aprovado com nota normal.");
    }else{
        document.write("Aprovado com louvor.");
    }
}
</script>
</head>
<body>

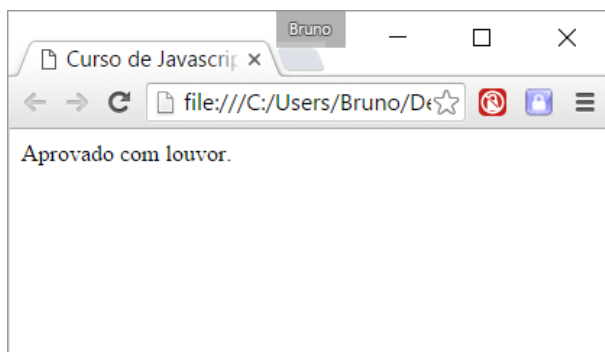
</body>
</html>
```

## Exemplo 2 – IF aninhado

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var nota=95;
      if(nota > 60){
        document.write("Aprovado ");
        if(nota < 90){
          document.write("com nota normal.");
        }else{
          document.write("com louvor.");
        }
      }else{
        document.write("Reprovado ");
        if(nota > 40){
          document.write("por nota baixa.");
        }else{
          document.write("com louvor.");
        }
      }
    </script>
  </head>
  <body>

  </body>
</html>
```

Para os códigos acima temos o resultado apresentado na ilustração a seguir.



## *Blocos de código, uso de { }*

Nas linguagens de programação é necessário uma forma de delimitar um bloco de código, determinar o início e o fim para que o sistema que esteja compilando ou interpretando o código consiga entender onde começa e termina uma determinada rotina.

Em javascript usamos as chaves { } para criar este limite, o uso é bem simples, veja alguns exemplos com ajuda do comando IF.

```
if(operação1){  
    comando 1;  
    comando 2;  
    comando 3;  
}
```

```
if(operação2){  
    comando 4;  
    comando 5;  
    comando 6;  
}
```

```
if(operação3){  
    comando 7;  
    comando 8;  
    comando 9;  
}
```

Como podemos determinar que os comandos 4, 5 e 6 pertencem ao segundo IF? Pelas chaves, porque os comandos 4, 5 e 6 estão dentro das chaves delimitadoras.

Para iniciar um bloco de comandos basta abrir a chave { e para terminar este bloco de comandos basta fechar a chave }.

Uma condição importante a observar é que alguns casos, como IF, SWITCH, FOR, WHILE, TRY, não tem necessidade de uso de chaves { }, mas somente quando existir somente um comando a ser executado, veja os exemplos.

```
if(operação1)  
    comando 1;
```

```
if(operação2)  
    comando 2;
```

```
if(operação3)  
    comando 3;
```

Nestes casos não há necessidade de uso das chaves { } porque cada IF só tem um comando, mas podemos usar as chaves mesmo assim? Sim, podemos. Então os exemplos a seguir são perfeitamente válidos.

```
if(operação1){  
    comando 1;  
}
```

```
if(operação2){  
    comando 2;  
}
```

```
if(operação3){  
    comando 3;  
}
```

Neste material, mesmo que exista somente um comando, eu vou usar as chaves, isso porque facilita na visualização e entendimento do código.

Um detalhe importante é quanto ao local da chave de abertura, pode ser usada na mesma linha ou na linha seguinte.

```
if(operação1)  
{  
    comando 1;  
    comando 2;  
}
```

```
if(operação2)  
{  
    comando 3;  
    comando 4;  
}
```

```
if(operação3)  
{  
    comando 5;  
    comando 6;  
}
```

## Switch - case

Outro comando importante para decisão é o comando “switch”, neste caso testamos uma expressão e definimos valores possíveis para esta expressão (cases), cada “case” possui os comandos a serem executados, vamos ver a sintaxe de uso do comando.

```
switch(expressão) {  
    case valor 1:  
        //comandos  
        break;  
    case valor 2:  
        //comandos  
        break;  
    case valor 3:  
        //comandos  
        break;  
    default:  
        //comandos  
}
```

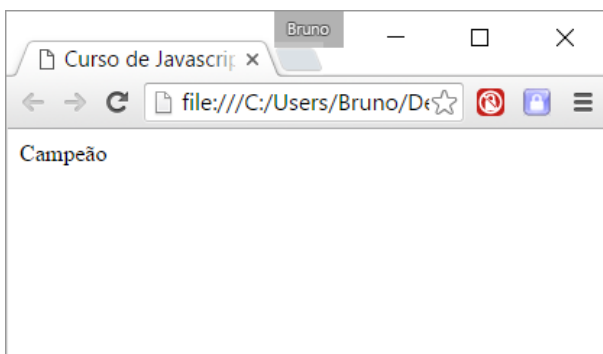
Nesta sintaxe cada case compara seu valor com a expressão, o que for equivalente, executa seus comandos, a última opção, default, será a opção válida caso nenhum dos cases anteriores tiver o valor igual à expressão, ou seja, se nenhum case for igual os comandos executados serão os comandos do default.

Vamos a um programa para verificar a posição de chegada de um determinado corredor e imprimir na tela o resultado.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var pos=1;
      switch(pos){
        case 1:
          document.write("Campeão");
          break;
        case 2:
          document.write("Segundo lugar");
          break;
        case 3:
          document.write("Terceiro lugar");
          break;
        default:
          document.write("Não subiu no pódio");
      }
    </script>
  </head>
  <body>

  </body>
</html>
```

Neste caso, como o valor da variável é igual a 1 o resultado será “Campeão” como podemos ver na ilustração a seguir.



Podemos “unir” vários cases para para executar um mesmo bloco de comandos, isso é bastante corriqueiro e fácil de programar, vamos mudar nosso programa para informar somente se o corredor conseguiu subir ao pódio ou não.

```
<script>
  var pos=1;
  switch(pos){
    case 1:
    case 2:
    case 3:
      document.write("Subiu ao pódio");
      break;
    default:
      document.write("Não subiu ao pódio");
  }
</script>
```

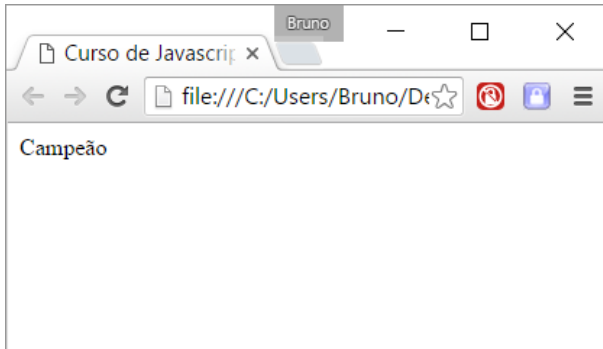
No código acima caso a posição seja 1, 2 ou 3 será mostrado o texto “Subiu ao pódio” qualquer valor diferente destes será mostrado o texto “Não subiu ao pódio”.

Podemos inserir quantos comandos forem necessários para um case, inclusive realizar testes com o comando IF, vamos incluir algumas instruções IF dentro do case, para fins de aprendizagem, obviamente que no código a seguir não se faz necessária a utilização deste IF, só quero mostrar que é perfeitamente possível utilizar IF dentro de switch.

```
<script>
  var pos=1;
  switch(pos){
    case 1:
```

```
case 2:
case 3:
    if(pos == 1){
        document.write("Campeão");
    }else if(pos == 2){
        document.write("Segundo lugar");
    }else{
        document.write("Terceiro lugar");
    }
    break;
default:
    document.write("Não subiu ao pódio");
}
</script>
```

O resultado é o mesmo do código anterior.



## Array

Arrays são estrutura bastante interessantes onde podemos “coleccionar” variáveis, isso mesmo, “coleccionar”.

Lembra quando você colecionava figurinhas? Se bem que os meninos de hoje quase não colecionam mais figurinhas! Acho que tô ficando velho?!?

Pois bem, é basicamente uma “coleção” de variáveis, como um container que irá armazenar variáveis e em Javascript estes valores não precisam ser do mesmo tipo como em linguagens tradicionais como C++, essa é mais uma das particularidades e facilidades de Javascript, como não precisamos informar o tipo de dados, em um mesmo array, podemos armazenar int, string, float, bool, etc.

Por exemplo, em um jogo você pode criar um array chamado mochila que irá armazenar todos os itens que o jogador carrega, pode criar um array de inimigos e por ai vai.

A ideia é simples, sempre que você precise criar várias variáveis do mesmo tipo, pode usar o array para facilitar o uso.

Os Arrays podem ser unidimensionais, bidimensionais ou multidimensionais, veja a seguir a sintaxe básica para se criar um array unidimensional.

```
var nomeArray=new Array();
```

Acima temos o código “básico” para se declarar um array unidimensional, neste código básico não informamos tamanho, simplesmente que precisamos criar essa coleção de valores chamada “nomeArray”.

Vamos entender a seguir como trabalhar com os arrays.

### Unidimensional / Vetor

Um array unidimensional basicamente é um array com somente um índice de controle, com uma dimensão, para entender melhor veja a ilustração a seguir que mostra um array com 5 posições, cada posição com um valor “string”

var mochila=new Array()

corda	faca	arame	lanterna	pedra
0	1	2	3	4

Vou mostrar o código para podermos criar este array.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var mochila=new Array();

      mochila[0]="corda";
      mochila[1]="faca";
      mochila[2]="arame";
      mochila[3]="lanterna";
      mochila[4]="pedra";
    </script>
  </head>
  <body>

  </body>
</html>
```

De acordo com a ilustração e o código anteriores, vimos que nosso array/vetor tem 5 posições, ou seja, tamanho 5.

Para indicar uma posição, usamos um valor entre os colchetes [], vamos nos referir a esse valor como “índice”, ou seja, [3] indica que estamos na posição de índice 3 do array. O que não podemos esquecer é que a primeira posição tem índice zero [0] e não [1], o elemento que tem índice [1] está na segunda posição.

Sendo assim, de acordo com o código anterior, o elemento “lanterna” está na quarta posição, de índice [3].

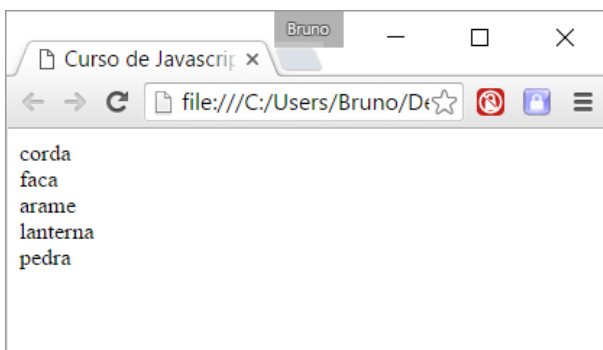
Vamos imprimir os elementos do nosso array, como ainda não aprendemos sobre loops, vamos imprimir um a um.

```
<script>
  var mochila=new Array();

  mochila[0]="corda";
  mochila[1]="faca";
  mochila[2]="arame";
  mochila[3]="lanterna";
  mochila[4]="pedra";

  document.write(mochila[0]+"<br>");
  document.write(mochila[1]+"<br>");
  document.write(mochila[2]+"<br>");
  document.write(mochila[3]+"<br>");
  document.write(mochila[4]+"<br>");
</script>
```

O resultado deste código pode ser visto na ilustração ao seguir.



Existem alguns métodos que podemos utilizar para melhorar o trabalho com os arrays, vou mostrar uma tabela com alguns métodos interessantes.

Método	Descrição	Exemplo
concat()	Junta os elementos de dois ou mais arrays e retorna uma cópia com os elementos juntos	var ar1 = ["Carro", "Moto"]; var ar2 = ["Navio", "Patins", "Skate"]; var transp = ar1.concat(ar2);
indexOf()	Procura por um elemento específico no array e retorna a sua posição	var cores = ["Verde", "Azul", "Amarelo"]; var pos = cores.indexOf("Azul"); //Retorna 1
join()	Junta todos os elementos de um array em uma string	var cores = ["Verde", "Azul", "Amarelo"]; var paleta = cores.join();
push()	Insere um novo elemento no final do array	var cores = ["Verde", "Azul", "Amarelo"]; cores.push("Vermelho");
pop()	Remove o último elemento do array	var cores = ["Verde", "Azul", "Amarelo"]; cores.pop();
reverse()	Inverte a ordem dos elementos do array	cores.reverse();
shift()	Remove o primeiro elemento do array	var cores = ["Verde", "Azul", "Amarelo"]; cores.shift();
sort()	Ordena os elementos do array em ordem crescente	cores.sort();
toString()	Converte um array em string e retorna essa string	var cores = ["Verde", "Azul", "Amarelo"]; cores.toString();
unshift()	Insere um novo elemento no início do array	var cores = ["Verde", "Azul", "Amarelo"]; cores.unshift("Vermelho");
splice()	Corta o array em um ponto indicado.	var num = [0,1,2,3,4,5,6,7,8,9]; num.splice(2,4); //Remove 4 elementos a partir do índice 2 //Resultado: [0,1,2,3,6,7,8,9]

Vamos usar alguns dos métodos para entender seu funcionamento.

1. No código a seguir vamos inserir cinco elementos no array usando o método "push".
2. Na sequência vamos remover o elemento de índice 2, ou seja, o terceiro elemento "arame".
3. Depois vamos ordenar os itens do array em ordem crescente usando o método "sort".
4. Vamos imprimir a quantidade de elementos no array com o "length", lembre-se que removemos um.
5. Por último vamos imprimir os itens da mochila, um a um.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var mochila=new Array();

      mochila.push("corda");
      mochila.push("faca");
      mochila.push("arame");
      mochila.push("lanterna");
      mochila.push("pedra");

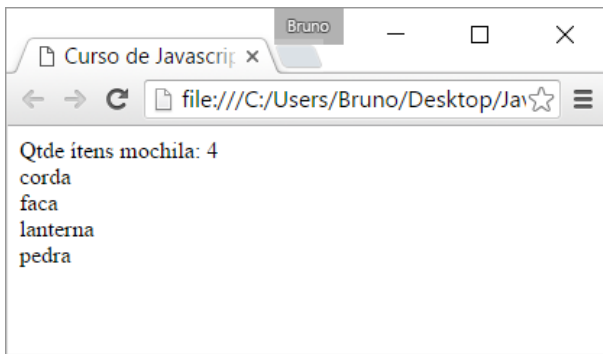
      mochila.splice(2,1);
      mochila.sort();
      document.write("Qtde itens mochila: " + mochila.length + "<br>");

      document.write(mochila[0]+"<br>");
      document.write(mochila[1]+"<br>");
      document.write(mochila[2]+"<br>");
      document.write(mochila[3]+"<br>");
    </script>
  </head>
  <body>

</body>
</html>
```

Confira a imagem a seguir que mostra o resultado do programa acima.





As possibilidades de uso para arrays são imensas, podemos utilizar para armazenar qualquer tipo de informações e manipular estas informações com muita facilidade.

Uma outra maneira de declarar um array já adicionando valores dentro dele é a seguinte:

```
var nomeArray=[item1, item2, item3];
```

Veja o exemplo a seguir onde declaro um array com 5 itens.

```
<script>
    var mochila=["corda","faca","arame","lanterna","pedra"];

    document.write(mochila[0]+"<br>");
    document.write(mochila[1]+"<br>");
    document.write(mochila[2]+"<br>");
    document.write(mochila[3]+"<br>");
    document.write(mochila[4]+"<br>");
</script>
```

Podemos usar a mesma maneira anterior, porém, inserindo os valores após a declaração, como no código a seguir.

```
<script>
    var mochila=[];

    mochila[0]="corda";
    mochila[1]="faca";
    mochila[2]="arame";
    mochila[3]="lanterna";
    mochila[4]="pedra";

    document.write(mochila[0]+"<br>");
    document.write(mochila[1]+"<br>");
    document.write(mochila[2]+"<br>");
    document.write(mochila[3]+"<br>");
    document.write(mochila[4]+"<br>");
</script>
```

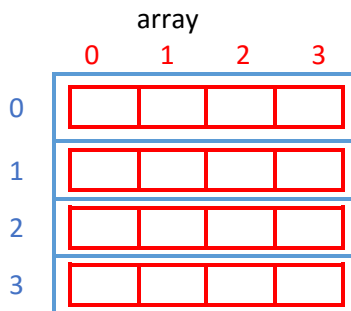
## ***Bidimensional / Matriz***

Anteriormente vimos os arrays com índices únicos, unidimensionais, como se estivéssemos uma tabela com somente uma linha ou coluna para guardar as informações.

Nesta parte do curso iremos aprender sobre os arrays bidimensionais, imagine uma coleção de arrays, ou simplesmente imagine nosso array unidimensional em uma coluna como a ilustração a seguir.

array	
0	<input type="text"/>
1	<input type="text"/>
2	<input type="text"/>
3	<input type="text"/>
4	<input type="text"/>
5	<input type="text"/>

Imagine agora que para cada posição deste array iremos adicionar um novo array em linha como na ilustração a seguir.



Este é um array bidimensional e como pode ser visto na ilustração acima possui dois índices para controle um para “linhas” outro para “colunas”.

Implementar esse tipo de Array em Javascript não é tão trivial quanto em outras linguagens, aqui, precisamos literalmente criar um array dentro de outro array, no nosso exemplo da mochila, precisamos criar um array para os itens e adicionar cada um destes itens/arrays em uma posição de outro array.

Vamos ver este código.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var mochila=new Array();

      var item1=["corda",1];
      var item2=["faca",2];
      var item3=["remedio",10];
      var item4=["pedra",32];
      var item5=["pilha",6];
      var item6=["lanterna",1];

      mochila[0]=item1;
      mochila[1]=item2;
      mochila[2]=item3;
      mochila[3]=item4;
      mochila[4]=item5;
      mochila[5]=item6;

      document.write(mochila[0][0]+": "+mochila[0][1]+"<br>");
      document.write(mochila[1][0]+": "+mochila[1][1]+"<br>");
      document.write(mochila[2][0]+": "+mochila[2][1]+"<br>");
      document.write(mochila[3][0]+": "+mochila[3][1]+"<br>");
      document.write(mochila[4][0]+": "+mochila[4][1]+"<br>");
      document.write(mochila[5][0]+": "+mochila[5][1]+"<br>");
    </script>
  </head>
  <body>

  </body>
</html>
```

No código acima criamos um array chamado “mochila”, na sequência criamos 6 arrays (item1 a item6) com duas posições cada, onde guardamos o nome do item e a quantidade deste item na mochila.

O próximo passo foi adicionar cada um destes itens em uma posição da mochila e por último imprimimos os itens e suas quantidades na tela.

Pra facilitar a visualização veja a ilustração a seguir.

	mochila / item1 ao item6	
	0	1
0	"corda"	1
1	"faca"	2
2	"remedio"	10
3	"pedra"	32
4	"pilha"	6
5	"lanterna"	1

Então, para acessarmos os elementos deste array bidimensional ou matriz, precisamos informar um índice para o array mochila e outro para os arrays dos itens, desta forma `mochila[indice1][indice2]`.

Veja os comandos para impressão em nosso código anterior.

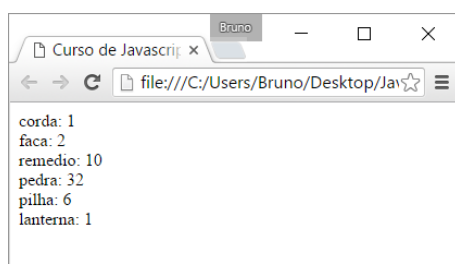
```
document.write(mochila[0][0]+": "+mochila[0][1]+"<br>");
```

Neste comando imprimimos o elemento "corda" posicionado na linha [0] coluna [0], ou seja, posição [0] do array "mochila" e posição [0] do array "item1".

Depois imprimimos o elemento que está posição [0] do array "mochila" e posição [1] do array "item1" que resulta na impressão do numeral 1.

```
mochila[0][0] → corda
mochila[1][0] → faca
mochila[2][0] → remédio
mochila[3][0] → pedra
mochila[4][0] → pilha
mochila[5][0] → lanterna
mochila[0][1] → 1
mochila[1][1] → 2
mochila[2][1] → 10
mochila[3][1] → 32
mochila[4][1] → 6
mochila[5][1] → 1
```

O resultado do código anterior é a ilustração a seguir.



Ainda podemos facilitar um pouco mais e usar somente um vetor “item”, basta modificar seus valores antes de inserir no vetor mochila, veja o código a seguir.

```
<script>
    var mochila=new Array();
    var item;

    item=["corda",1];
    mochila[0]=item;

    item=["faca",2];
    mochila[1]=item;

    item=["remedio",10];
    mochila[2]=item;

    item=["pedra",32];
    mochila[3]=item;

    item=["pilha",6];
    mochila[4]=item;

    item=["lanterna",1];
    mochila[5]=item;

    document.write(mochila[0][0]+": "+mochila[0][1]+"<br>");
    document.write(mochila[1][0]+": "+mochila[1][1]+"<br>");
    document.write(mochila[2][0]+": "+mochila[2][1]+"<br>");
    document.write(mochila[3][0]+": "+mochila[3][1]+"<br>");
    document.write(mochila[4][0]+": "+mochila[4][1]+"<br>");
    document.write(mochila[5][0]+": "+mochila[5][1]+"<br>");
</script>
```

Assim como fazemos com arrays unidimensionais podemos declarar os arrays bidimensionais e já adicionar valores, este procedimento vai nos poupar um pouco de código, confira o código a seguir.

```
<script>
    var mochila=[["corda",1],["faca",2],["remedio",10],["pedra",32],["pilha",6],["lanterna",1]];

    document.write(mochila[0][0]+": "+mochila[0][1]+"<br>");
    document.write(mochila[1][0]+": "+mochila[1][1]+"<br>");
    document.write(mochila[2][0]+": "+mochila[2][1]+"<br>");
    document.write(mochila[3][0]+": "+mochila[3][1]+"<br>");
    document.write(mochila[4][0]+": "+mochila[4][1]+"<br>");
    document.write(mochila[5][0]+": "+mochila[5][1]+"<br>");
</script>
```

O resultado é exatamente o mesmo em ambos os códigos acima.

## *Incremento e decremento de variáveis*

Muitas vezes precisaremos aumentar ou diminuir o valor de uma variável e existem formas que simplificam muito este trabalho, é o que veremos nesta parte do curso.

Vamos criar um código que faz um incremento padrão de um em uma variável.

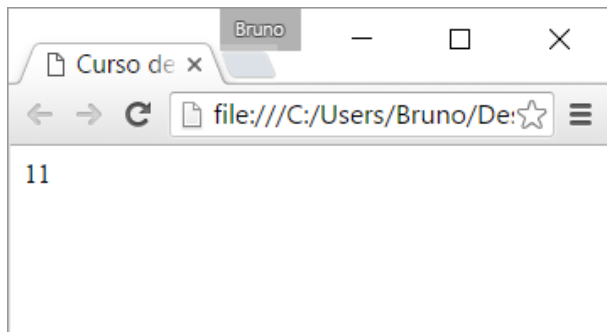
```
<!doctype html>
<html lang="pt-br">
    <head>
        <title>Curso de Javascript</title>
        <meta charset="UTF-8">
        <script>
            var num=10;
            num=num+1;
            document.write(num);
        </script>
    </head>
    <body>

    </body>
</html>
```

Primeiramente criamos a variável num com valor inicial 10.

Na sequência incrementamos o valor de num em 1 e escrevemos esse valor na tela.

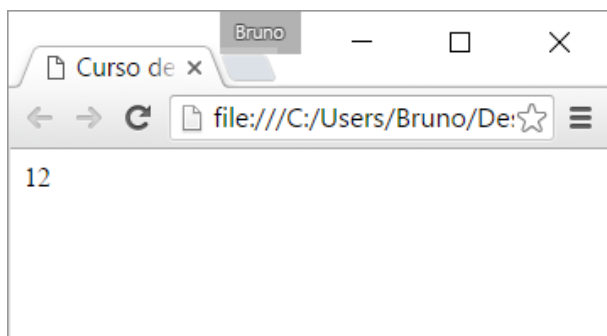
Este incremento padrão simplesmente adiciona a variável num seu valor que é 10 somado a um, então num passa a ter o valor 11.



Para facilitar esta operação podemos usar duas outras formas, veja o código a seguir.

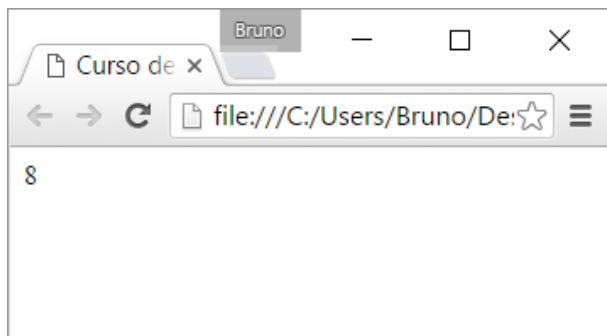
```
<script>
  var num=10;
  num+=1;
  num++;
  document.write(num);
</script>
```

Os comandos “num=num+1”, “num+=1” e “num++” são igual, resultam no mesmo fim.



De forma contrária podemos usar para decrementar/diminuir o valor da variável, veja o código.

```
<script>
  var num=10;
  num-=1;
  num--;
  document.write(num);
</script>
```



Os comandos num++ e num—vão ser usados sempre que precisarmos adicionar ou remover o valor de uma variável em 1.

Se precisar adicionar ou remover mais de 1? Então usamos os comandos a seguir.

Somar 5 ao valor da variável.

```
<script>
  var num=10;
  num+=5;
  document.write(num);
</script>
```

Subtrair 7 ao valor da variável.

```
<script>
  var num=10;
  num-=7;
  document.write(num);
</script>
```

Usando esta mesma técnica podemos usar com operadores de multiplicação e divisão.

Multiplica o valor da variável com 2.

```
<script>
  var num=10;
  num*=2;
  document.write(num);
</script>
```

Divide o valor da variável com 2.

```
<script>
  var num=10;
  num/=2;
  document.write(num);
</script>
```

## *Pós e Pré incremento*

Como vimos as operações de incremento `num++` e decremento `num--`, aumentam e diminuem o valor da variável em 1, mas existem ainda uma observação a ser considerada, o momento em que a variável é incrementada.

Então vamos discutir um pouco de pós e pré incremento, veja o código a seguir.

```
<script>
  var num=10;
  var x;
  x=num++;
  document.write(x);
</script>
```

Veja o resultado do código.



Note que embora incrementamos o valor de `num` e passamos para a variável `x`, ao imprimir não é impresso o valor 11 e sim o valor 10, mas como? Isso porque usamos pós incremento.

No pós incremento o valor é adicionado à variável depois e no pré incremento é adicionado antes, vamos ver nosso código novamente.

```
x=num++;
```

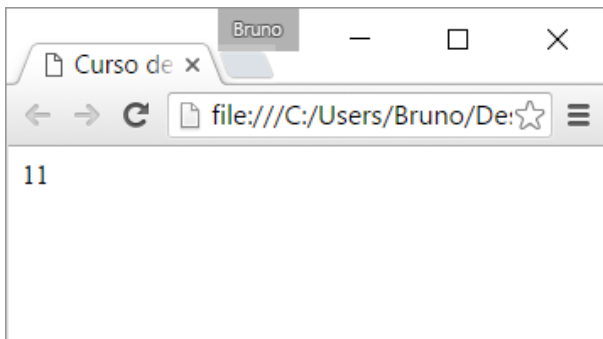
Por que o valor de X é 10 e não 11? Por que usamos pós incremento.

Neste caso primeiro é adicionado o valor de num a X e só depois num tem o valor incrementado, então, no momento da atribuição do valor de num a x, num ainda não foi incrementada.

Vamos alterar nosso código.

```
<script>
  var num=10;
  var x;
  x=++num;
  document.write(x);
</script>
```

Confira o resultado deste código.



Neste caso o comportamento do programa é diferente.

```
x=++num;
```

O incremento é feito antes que o valor seja passado à variável, isso é chamado de pré incremento.

## Loops / Estruturas de repetição

Em programação repetir um determinado comando ou rotina inteira com vários comandos é bem rotineiro, para facilitar essas repetições temos os comandos de loop, que repetem um bloco de comandos uma certa quantidade de vezes.

Aprender os comandos de loop e suas diferenças é fundamental em qualquer linguagem e neste capítulo vamos aprender como utilizar esses comandos.

### For / Para

O comando for é uma estrutura de repetição bastante versátil, vamos procurar usar esta estrutura quando estivermos certos do número de vezes que iremos repetir os comandos dentro do for, para inicializar ou percorrer vetores a estrutura for é extremamente indicada, porém, podemos fazer com outras estruturas também.

Vamos conferir a sintaxe do comando FOR.

for(inicialização do contador; condição de execução ou parada do loop; incremento/decremento do contador)

Vamos mostrar um código onde o loop repete um comando de escrita dez vezes.

```
<!doctype html>
<html lang="pt-br">
```

```
<head>
  <title>Curso de Javascript</title>
  <meta charset="UTF-8">
  <script>
    var i;
    for(i=0; i<10; i++){
      document.write("CFB<br>");
    }
  </script>
</head>
<body>

</body>
</html>
```

No código anterior declaramos uma variável “i” para ser usada como contador de loop no FOR.

Veja a leitura do comando FOR que acabamos de implementar.

**for(i=0; i<10; i++);**  
**para(i iniciando em 0; enquanto i for menor que 10; aumente o valor de i em 1)**

Neste FOR configuramos 10 iterações, ou seja, os comandos dentro das chaves {} serão executados 10 vezes, vou mostrar um passo a passo da execução deste FOR.

Inicia a variável “i” com valor 0.

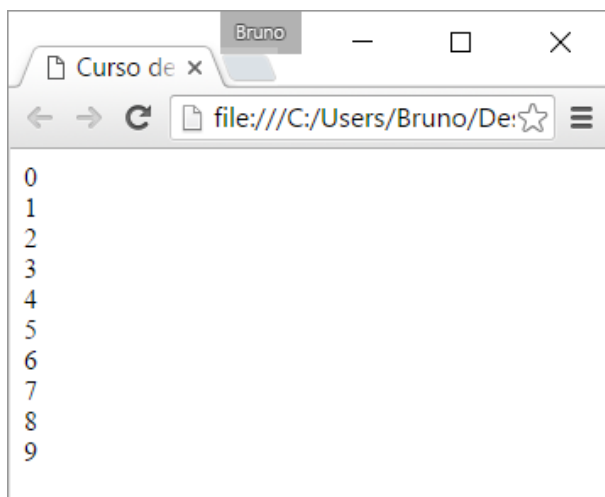
Verifica se o valor de “i” é menor que 10, se for (true) executa os comandos entre as chaves {}.

Aumenta o valor de i em 1.

Vamos fazer uma pequena alteração em nosso código para imprimir uma contagem de 0 a 9.

```
<script>
  var i;
  for(i=0; i<10; i++){
    document.write(i+"<br>");
  }
</script>
```

O resultado deste comando pode ser visto a seguir.



Outro ponto interessante é que podemos declarar diretamente o contador dentro do for, como no código a seguir.

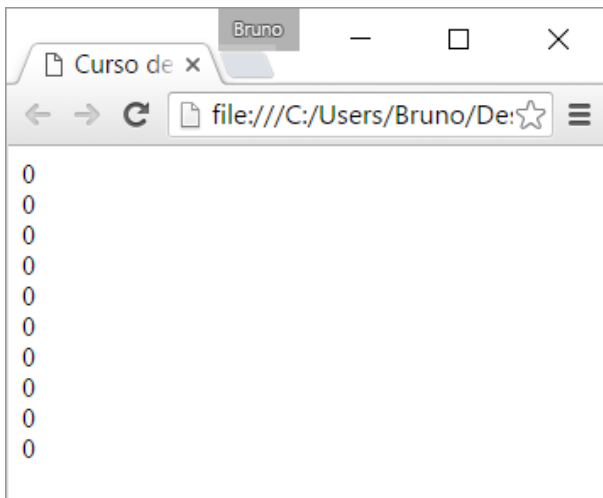
```
<script>
  for(var i=0; i<10; i++){
    document.write(i+"<br>");
  }
</script>
```



Como disse anteriormente é muito comum usarmos FOR para percorrer arrays, então vamos criar um código que declara um array e um loop FOR para preenche esse array com valores 0 e outro FOR para imprimir o conteúdo deste array.

```
<script>
    var num=new Array();
    var i;
    for(i=0; i<10; i++){
        num[i]=0;
    }
    for(i=0; i<10; i++){
        document.write(num[i]+"<br>");
    }
</script>
```

Aqui está o resultado do código.



Podemos realizar esta mesma ação usando um só vetor, veja a alteração a seguir.

```
<script>
    var num=new Array();
    var i;
    for(i=0; i<10; i++){
        num[i]=0;
        document.write(num[i]+"<br>");
    }
</script>
```

Para melhorar a relação do FOR com o array, ainda podemos usar o tamanho do array na condição de execução do FOR, veja a alteração a seguir.

```
<script>
    var num=new Array(10);
    var i;
    for(i=0; i<num.length; i++){
        num[i]=0;
        document.write(num[i]+"<br>");
    }
</script>
```

Note que neste caso, como o array ainda não tinha sido preenchido com nenhum valor, informamos o tamanho na declaração do array `var num=new Array(10);`

Como informamos o tamanho na declaração do array, neste caso, podemos usar o método “length” que retorna o tamanho do array.

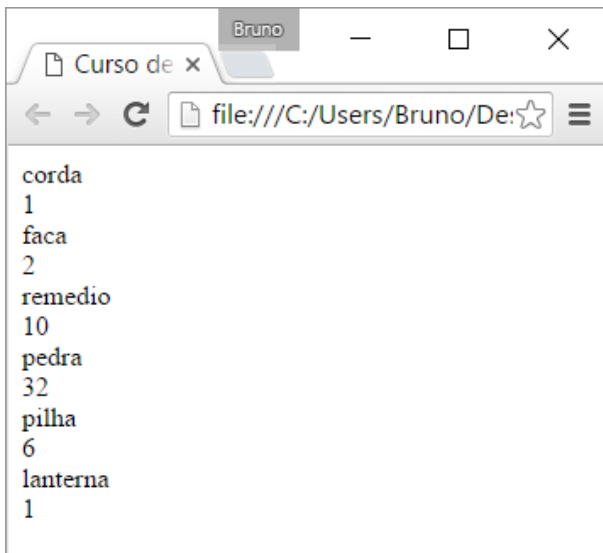
No caso de arrays bidimensionais precisamos usar um FOR dentro do outro, um FOR para controlar o primeiro array e um FOR para o segundo array.

Veja o código a seguir.

```
<script>
    var mochila=[["corda",1],["faca",2],["remedio",10],["pedra",32],["pilha",6],["lanterna",1]];

    for(var l=0;l<6;l++){
        for(var c=0;c<2;c++){
            document.write(mochila[l][c]+"<br>");
        }
    }
</script>
```

O resultado deste código é a imagem a seguir.

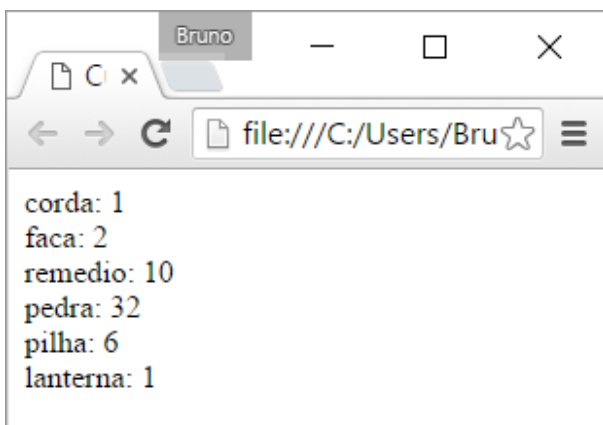


E se precisarmos imprimir a quantidade de itens ao lado do item? Neste caso podemos usar um único FOR.

```
<script>
    var mochila=[["corda",1],["faca",2],["remedio",10],["pedra",32],["pilha",6],["lanterna",1]];

    for(var i=0;i<6;i++){
        document.write(mochila[i][0]+": "+mochila[i][1]+"<br>");
    }
</script>
```

Veja o resultado deste código.



A este ponto você já deve ter entendido bem o comando FOR, vamos prosseguir a aprender outro comando para loop, o comando while.

## While / Enquanto

Outro comando de loop importante que precisamos aprender é o comando while, também serve para repetir comandos como o FOR, mas tem uma sintaxe diferente.

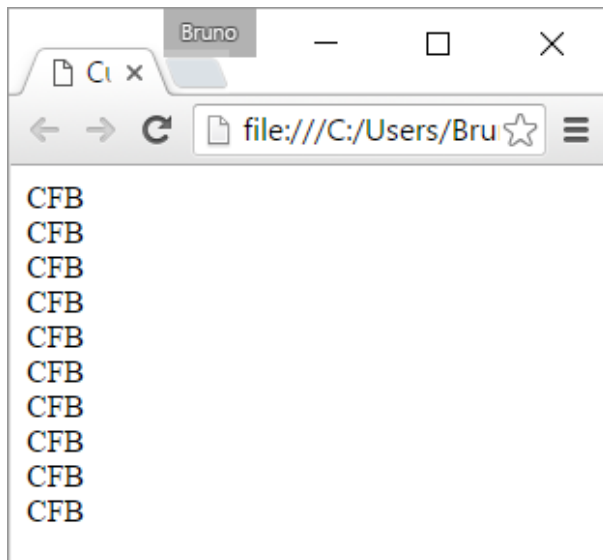
Inicialização do contador;

```
while( condição para execução ){  
    comandos;  
    incremento do contador;  
}
```

Vamos ver um programa básico usando o loop while.

```
<!doctype html>  
<html lang="pt-br">  
  <head>  
    <title>Curso de Javascript</title>  
    <meta charset="UTF-8">  
    <script>  
      var cont;  
  
      cont=0;  
      while(cont < 10){  
        document.write("CFB<br>");  
        cont++;  
      }  
    </script>  
  </head>  
  <body>  
  
  </body>  
</html>
```

Confira a seguir o resultado do código.



Veja a seguir como podemos ler o comando while para facilitar nosso entendimento.

```
cont=0;  
while(cont < 10){  
    document.write("CFB<br>");  
    cont++;  
}
```

```
Inicie o contador com o valor 0;
enquanto(contador for menor que 10){
    document.write("CFB<br>");
   Incremente o contador em 1;
}
```

Note que no loop while o incremento é feito dentro do bloco de comandos e a inicialização do “contador” de controle é feita fora, antes de entrar no loop while, veja a seguir dois loops FOR e WHILE que geram resultados semelhantes.

## FOR

```
<script>
    var cont;

    for (cont=0; cont<10; cont++) {
        document.write("CFB<br>");
    }
</script>
```

## WHILE

```
<script>
    var cont;

    cont=0;
    while (cont < 10) {
        document.write("CFB<br>");
        cont++;
    }
</script>
```

Um detalhe importante que precisamos ressaltar sobre o while é que se a condição de parada já for satisfeita na primeira execução do while, os comandos entre as chaves {} não serão executados, veja o código a seguir.

```
<script>
    var cont;

    cont=25;
    while (cont<10) {
        document.write("CFB<br>");
        cont++;
    }
</script>
```

Teste o código acima e veja que não será impresso nada na tela, pois, a condição para execução já foi satisfeita ao iniciar o while. Para resolver esse caso precisamos usar o loop “Do While”.

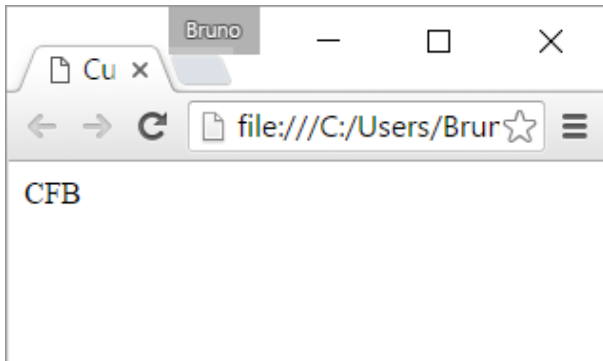
## *Do While / Faça Enquanto*

O loop “do while” se diferencia do loop “while” pelo momento do teste, enquanto o loop while testa a condição de execução no início do loop e só depois executa os comandos, no “do while” o teste ocorre no fim, primeiro são executados os comandos do loop e só no final é feito o teste, veja o código a seguir.

```
<script>
    var cont;

    cont=25;
    do{
        document.write("CFB<br>");
        cont++;
    }while (cont<10);
</script>
```

Note que a variável `cont` é iniciada com o valor 25, o que faz com que a condição de execução do loop já seja satisfeita `while(cont<10)`, porém, mesmo assim os comandos do loop `{}` são executados pelo menos uma vez, veja a ilustração a seguir que mostra o resultado do código.



Então, a principal diferença entre “while” e “do while” é que o loop “do while” garante a execução dos comandos pelo menos uma vez, mesmo que a condição de execução já tenha sido satisfeita, o que não acontece no “while”.

### *Loops infinitos – cuidado*

Um cuidado extremo que devemos tomar com os loops é não torna-los loops infinitos, o que é isso? É simplesmente um loop onde a condição de parada nunca é satisfeita e o loop é executado sem parar.

Vamos ver loops infinitos no FOR, WHILE e DO WHILE.

#### FOR infinito

```
<script>
  var i;

  for(i=15; i>10; i++){
    document.write("CFB<br>");
  }
</script>
```

Note que a variável “i” é iniciada com 15 e a condição de execução é `(i>10)` como estamos incrementando o valor de “i++” ele sempre será maior que 10.

#### WHILE infinito

```
<script>
  var i;

  i=15;
  while(i>10){
    document.write("CFB<br>");
    i++;
  }
</script>
```

Este while infinito é o mesmo caso do FOR.

#### DO WHILE infinito

```
<script>
  var i;

  i=15;
  do{
    document.write("CFB<br>");
    i++;
  }while(i>10);
</script>
```

Outro cuidado importante que devemos tomar é quanto ao incremento dentro do WHILE e DO WHILE, caso esqueçamos de realizar este controle o loop também se torna um loop infinito.

## Try – Catch – Finally

Neste capítulo iremos aprender a tratar exceções, basicamente iremos aprender a tratar erros, a instrução try “monitora” a execução dos comandos do bloco, caso haja algum erro, este erro é passado à instrução catch que executa seu bloco de comandos, caso não ocorra nenhum erro o bloco catch é ignorado, ao final são executados os comandos do bloco finally.

Em resumo, try executa um ou mais comandos, caso haja algum erro nesta execução o bloco catch é executado, no final de tudo o bloco finally é executado.

Vamos ver um código de exemplo, o comando em vermelho foi escrito de forma incorreta de propósito, para que seja gerado um erro em try.

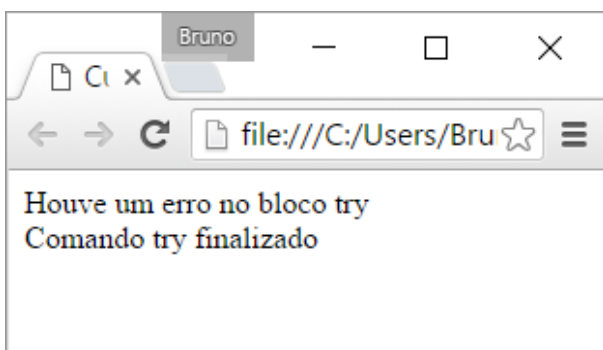
```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      try{
        alertt("CFB");
      }catch(erro){
        document.write("Houve um erro no bloco try<br>");
      }finally{
        document.write("Comando try finalizado<br>");
      }
    </script>
  </head>
  <body>

  </body>
</html>
```

Neste bloco try será gerado um erro, pois, o comando alert está escrito de forma incorreta, então, a execução é passada ao bloco catch que executa o comando de impressão.

Ao final de todo processo o bloco finally é executado imprimindo a mensagem.

Veja o resultado do programa.

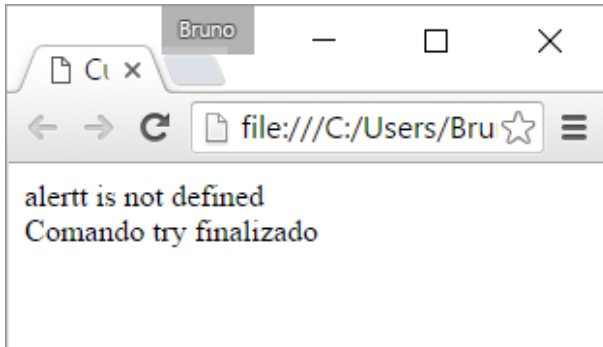


Podemos interceptar a mensagem de erro gerada pelo try, essa mensagem é passada ao catch pelo parâmetro que demos o nome de “erro” catch(erro), podemos usar outro nome para este parâmetro, mas o nome “erro” é bem sugestivo.

Vamos alterar nosso programa para interceptar a mensagem de erro e mostra-la na tela, altere o programa conforme o código destacado em vermelho.

```
<script>
  try{
    alertt("CFB");
  }catch(erro){
    document.write(erro.message+"<br>");
  }finally{
    document.write("Comando try finalizado<br>");
  }
</script>
```

Veja o resultado deste código.



Vamos alterar o código novamente de forma que não gere erro no comando try.

```
<script>
  try{
    alert("CFB");
  }catch(erro){
    document.write("Houve um erro no bloco try<br>");
  }finally{
    document.write("Comando try finalizado<br>");
  }
</script>
```

Com este código o comando alert será executado normalmente e não será gerado nenhum erro, então o bloco catch não é executado.

Um detalhe importante é que o bloco finally não é obrigatório, caso não queira executar comandos ao finalizar a instrução try basta não utilizar o bloco finally, como no código a seguir.

```
<script>
  try{
    alert("CFB");
  }catch(erro){
    document.write("Houve um erro no bloco try<br>");
    document.write("Erro gerado: "+erro.message+"<br>");
  }
</script>
```

Em alguns momentos será indispensável o uso de try ao executar um comando, simplesmente para podermos tratar o erro, caso, ocorra de forma mais racional.

## getElementById

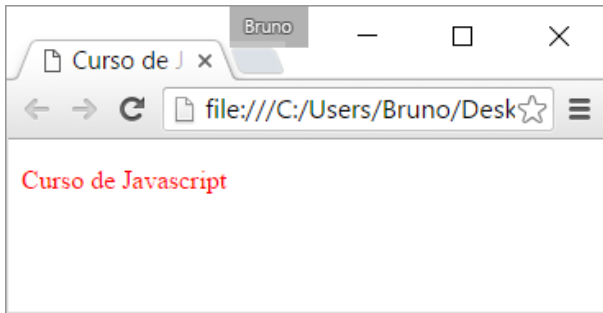
É muito comum precisarmos nos referenciar a um elemento HTML, para alterar um estilo CSS por exemplo, para isto temos à nossa disposição o método "getElementById", vamos ver um exemplo de uso.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var tx;
    </script>
```

```
</head>
<body>
  <p id="texto">Curso de Javascript</p>

  <script>
    tx=document.getElementById("texto");
    tx.style.color="#F00";
  </script>
</body>
</html>
```

No código acima relacionamos a tag <p> que tem o id="texto" com a variável "tx", em seguida alteramos a cor do texto para vermelho, veja o resultado.



## innerHTML

Finalmente pensaram em uma maneira de alterar o conteúdo de uma tag HTML de forma fácil. Você vai conhecer agora uma das melhores adições em javascript, acompanhe a explicação.

Vamos descrever uma tag <p> com um texto inicial.

```
<p id="texto">Curso de Javascript</p>
```

Suponhamos que em algum momento seja preciso mudar o texto definido por esse parágrafo, como fazer isso? Usando a propriedade "innerHTML", veja como usar esta propriedade para alterar o texto.

```
document.getElementById("texto").innerHTML="Canal Fessor Bruno";
```

Veja o código completo.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

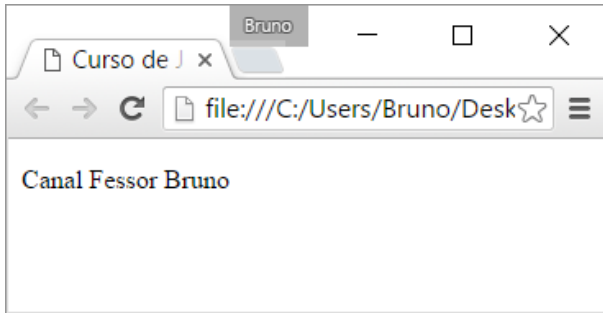
    </script>
  </head>
  <body>
    <p id="texto">Curso de Javascript</p>

    <script>
      document.getElementById("texto").innerHTML="Canal Fessor Bruno";
    </script>
  </body>
</html>
```

Originalmente o parágrafo com id="texto" possui o texto "Curso de Javascript", mas logo a seguir inserimos um script para mudar o conteúdo desta tag para "Canal Fessor Bruno".

O resultado deste programa é mostrado na ilustração a seguir.





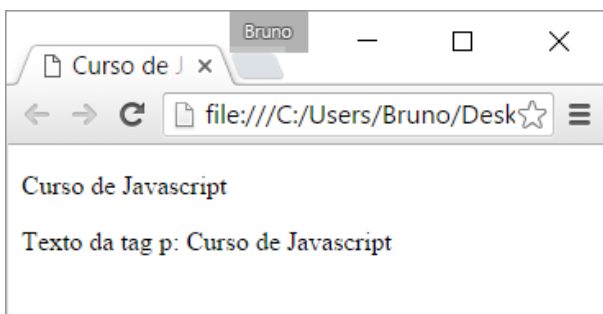
Podemos usar a propriedade “innerHTML” para simplesmente obter o valor de um elemento ou para alterar como já vimos, veja a seguir o procedimento necessário para pegar o texto de uma tag <p> e adicionar em uma variável.

Pegar o valor da tag e adicionar em uma variável.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var tx;
    </script>
  </head>
  <body>
    <p id="texto">Curso de Javascript</p>

    <script>
      tx=document.getElementById("texto").innerHTML;
      document.write("Texto da tag p: " + tx);
    </script>
  </body>
</html>
```

Veja o resultado deste código.



## getElementsByTagName

O método “getElementsByTagName” é um ótimo método para nos referenciar aos elementos HTML, este método retorna o(s) elemento(s) com o nome indicado, o ponto mais interessante é que caso haja várias tags iguais e isso é bastante comum, o método funciona como um vetor com todas as tags, por exemplo, se houver 10 tags <p> essas tags podem ser obtidas por array, vamos ver o código de exemplo.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var tagP;
    </script>
  </head>
  <body>
    <p>Canal Fessor Bruno</p>
```

```
<p>Curso de Javascript</p>
<p>www.cfbcursos.com.br</p>
<p>www.youtube.com.br</p>

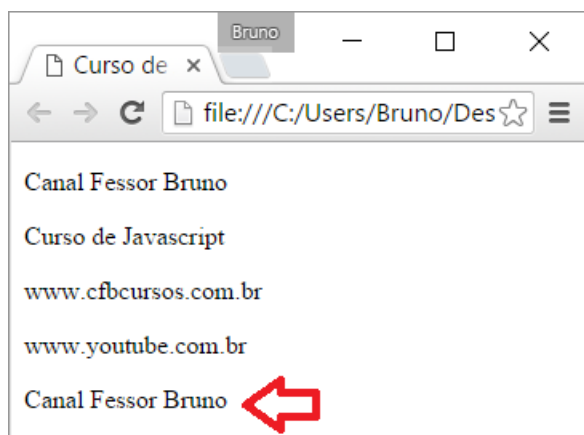
<script>
    tagP=document.getElementsByTagName("p")[0];
    document.write(tagP.innerHTML);
</script>
</body>
</html>
```

No código acima inserimos quatro tags <p> e no script usamos o método `getElementsByTagName` para pegar as tags <p>, note que no final do comando existe uma referência de arrays [0] indicando que queremos obter a primeira tag <p>, se for a segunda usamos o índice [1], a terceira o índice [2] e assim por diante.

`tagP=document.getElementsByTagName("p")[0];` → Retorna a primeira tag <p> e armazena na variável “tagP”.

`document.write(tagP.innerHTML);` → Aqui usamos a propriedade `innerHTML` da variável “tagP” para imprimir o texto desta tag <p>.

Confira a seguir o resultado do código acima, a seta vermelha indica o texto que foi impresso pelo javascript.



O vetor também pode ser referenciado na variável “tagP”, o resultado é o mesmo, veja o código a seguir.

```
<script>
    tagP=document.getElementsByTagName("p");
    document.write(tagP[0].innerHTML);
</script>
```

Poderíamos usar também diretamente em `document.write` como neste exemplo que o resultado também será o mesmo, `document.write(document.getElementsByTagName("p")[0].innerHTML);`

## querySelectorAll

[http://www.w3schools.com/jsref/met\\_document\\_queryselectorall.asp](http://www.w3schools.com/jsref/met_document_queryselectorall.asp)

O `querySelectorAll()` é um método que retorna os elementos do documento que correspondem a uma classe de CSS, como um objeto dentro de uma lista de objetos (array).

Uma das grandes vantagens de usar o método `querySelectorAll()` é que podemos nos referenciar tanto a tags como classes CSS.

Vamos ver alguns exemplos de uso do método.

**1** – Obter todos os elementos com a tag <p>, aqui vamos armazenar os elementos no array ps.

```
var ps = document.querySelectorAll("p");  
//Formata a cor de fundo para vermelho do primeiro elemento com a tag <p>.  
ps[0].style.backgroundColor="#F00";
```

**2** – Obter todos os elementos com a tag <p> que usam a classe aula, vamos armazenar os elementos no array ps.

```
var ps = document.querySelectorAll("p.aula");  
//Formata a cor de fundo para azul do primeiro elemento com a tag <p> que usa a classe aula.  
ps[0].style.backgroundColor="#00F";
```

**3** – Obter quantos elementos com a tag <p> que usam a classe aula, a quantidade é armazenada na variável qtdeps.

```
var qtdeps = document.querySelectorAll(".aula").length;
```

**4** – Obter todos os elementos que usam a classe aula, vamos armazenar os elementos no array ca.

```
var ca = document.querySelectorAll(".aula");  
//Formatar todos estes elementos a cor de fundo para verde.  
var i;  
for (i = 0; i < ca.length; i++) {  
    ca[i].style.backgroundColor="#0F0";  
}
```

**5** – Obter todos os elementos com a tag <p>, vamos armazenar os elementos no array ps.

```
var ps = document.querySelectorAll("p");  
//Formatar todos estes elementos a cor de fundo para verde.  
var i;  
for (i = 0; i < ps.length; i++) {  
    ps[i].style.backgroundColor="#0F0";  
}
```

**6** – Obter todos os elementos com a tag <a> que possuem a propriedade “target” configurada, vamos armazenar os elementos no array as.

```
var as = document.querySelectorAll("a[target]");  
//Formatar a borda de todos estes elementos com largura 2px, linha sólida e cor preto.  
var i;  
for (i = 0; i < as.length; i++) {  
    as[i].style.border="2px solid #000";  
}
```

**7** – Obter todos os elementos com a tag <p> que estão dentro de uma <div>, vamos armazenar os elementos no array pdv.

```
var pdv = document.querySelectorAll("div > p");  
//Formatar a cor de fundo destes elementos com a cor amarelo.  
var i;  
for (i = 0; i < pdv.length; i++) {  
    pdv[i].style.backgroundColor="#FF0";  
}
```

**8** – Obter todos os elementos com as tags <h1>, <div> e <span>, vamos armazenar os elementos no array tags.

```
var tags = document.querySelectorAll("h1, div, span");
//Formatar a cor de fundo destes elementos com a cor ciano.
var i;
for (i = 0; i < tags.length; i++) {
    tags[i].style.backgroundColor="#0FF";
}
```

Aprendemos anteriormente que com o método “querySelectorAll” podemos obter os elementos através das propriedades CSS, confira a tabela com as possibilidades que podemos usar para indicar um elemento com CSS ou uma tag usando “querySelectorAll”.

Seletor CSS ou tag	Exemplo	Descrição
<u>.class</u>	.intro	Seleciona todos os elementos que usam a classe intro / class="intro"
<u>#id</u>	#primeiroNome	Seleciona o elemento com o id primeiroNome / id="primeiroNome"
<u>*</u>	*	Seleciona todos os elementos
<u>elemento</u>	p	Seleciona todos os elementos com a tag <p>
<u>elemento elemento</u>	div, p	Seleciona todos os elementos <div> e <p>
<u>elemento elemento</u>	div p	Seleciona todos os elementos <p> que estão dentro de elementos <div>
<u>elemento&gt;elemento</u>	div > p	Seleciona todos os elementos <p> onde o pai seja um elemento <div>
<u>elemento+elemento</u>	div + p	Seleciona todos os elementos <p> que estão posicionados imediatamente após o elemento <div>
<u>elemento1~elemento2</u>	p ~ ul	Seleciona todos os elementos <ul> que são precedidos por um elemento <p>
<u>[atributo]</u>	[target]	Seleciona todos os elementos com o atributo target
<u>[atributo=valor]</u>	[target=_blank]	Seleciona todos os elementos com o atributo target configurado em _blank / target="_blank"
<u>[atributo~=valor]</u>	[title~=curso]	Seleciona todos os elementos cujo o atributo title contenha a palavra "curso"
<u>[atributo =valor]</u>	[lang =pt]	Seleciona todos os elementos com um valor de atributo lang começando com "pt"
<u>[atributo^=valor]</u>	a[href^="https"]	Seleciona cada elemento <a> cujo valor do atributo href começa com "https"
<u>[atributo\$=valor]</u>	a[href\$=".rar"]	Seleciona cada elemento <a> cujo valor do atributo href termina com ".rar"
<u>[atributo*=valor]</u>	a[href*="cursos"]	Seleciona a cada <uma> elemento cujo valor do atributo href contenha a substring "cursos"
<u>:active</u>	a:active	Seleciona o link ativo
<u>::after</u>	p::after	Insira algo depois do conteúdo de cada elemento <p>
<u>::before</u>	p::before	Insira algo antes do conteúdo de cada elemento <p>
<u>:checked</u>	input:checked	Seleciona todos os elementos <input> que tem a propriedade checked
<u>:disabled</u>	input:disabled	Seleciona todos os elementos <input> que tem a propriedade disabled
<u>:empty</u>	p:empty	Seleciona todos os elementos <p> que não tem filhos (incluindo os nós de texto)
<u>:enabled</u>	input:enabled	Seleciona todos os elementos <input> que tem a propriedade enabled
<u>:first-child</u>	p:first-child	Seleciona todos os elementos <p> que são first-child de seu elemento pai
<u>::first-letter</u>	p::first-letter	Seleciona a primeira letra de cada elemento <p>
<u>::first-line</u>	p::first-line	Selects the first line of every <p> element
<u>:first-of-type</u>	p:first-of-type	Seleciona a primeira linha de cada elemento <p>
<u>:focus</u>	input:focus	Seleciona o elemento de entrada que tem o foco
<u>:hover</u>	a:hover	Selecione os links que o tem a propriedade hover configurada
<u>:in-range</u>	input:in-range	Seleciona elementos de entrada com um valor dentro de um intervalo especificado
<u>:invalid</u>	input:invalid	Seleciona todos os elementos <input> que tem a propriedade invalid
<u>:lang(language)</u>	p:lang(en)	Seleciona todos os elementos <p> com um atributo lang igual a "en" (inglês)
<u>:last-child</u>	p:last-child	Seleciona todos os elementos <p> que é o último filho do seu elemento pai
<u>:last-of-type</u>	p:last-of-type	Seleciona todos os elementos <p> que é o último elemento <p> de seu elemento pai
<u>:link</u>	a:link	Seleciona todos os links não clicados
<u>:not(selector)</u>	:not(p)	Seleciona todos os elementos que não são um elemento <p>
<u>:nth-child(n)</u>	p:nth-child(2)	Seleciona o elemento <p> que é o segundo filho de seu elemento pai
<u>:nth-last-child(n)</u>	p:nth-last-child(2)	Seleciona o elemento <p> que é o segundo filho de seu elemento pai, a contar do último elemento filho
<u>:nth-last-of-type(n)</u>	p:nth-last-of-type(2)	Seleciona o elemento <p> que é o segundo elemento <p> de seu elemento pai, a contar do último elemento filho
<u>:nth-of-type(n)</u>	p:nth-of-type(2)	Seleciona o elemento <p> que é o segundo elemento <p> de seu elemento pai
<u>:only-of-type</u>	p:only-of-type	Seleciona todos os elementos <p> de seu elemento pai
<u>:only-child</u>	p:only-child	Seleciona todos os elementos <p> que só tem um elemento como filho
<u>:optional</u>	input:optional	Seleciona elementos input com nenhum atributo "required"
<u>:out-of-range</u>	input:out-of-range	Seleciona elementos input com um valor fora de um intervalo especificado
<u>:read-only</u>	input:read-only	Seleciona elementos input com o atributo "readonly" especificado
<u>:read-write</u>	input:read-write	Seleciona elementos input com o atributo "readonly" NÃO especificado
<u>:required</u>	input:required	Seleciona elementos input com o atributo "required" especificado
<u>:root</u>	:root	Seleciona o elemento raiz do documento
<u>::selection</u>	::selection	Seleciona a parte de um elemento que é selecionado pelo usuário
<u>:target</u>	#noticias:target	Seleciona o elemento #noticias atual (clicou em um URL que contém esse nome de âncora)
<u>:valid</u>	input:valid	Seleciona todos os elementos input com um valor válido
<u>:visited</u>	a:visited	Seleciona todos os links visitados

## Acessando elementos de formulários

Para formulários existe uma maneira muito simples de acessarmos os elementos, no caso dos formulários os elementos do <form> são todos armazenados em um vetor, assim como todos os elementos do documento.

Vamos ver alguns exemplos.

`tag=document.forms["curso"];` → Este comando permite obter todos os elementos do formulário “curso” através da variável “tag”.

`document.write(tag.elements["fNome"].value);` → Com o vetor de elementos passado à variável “tag” agora podemos referenciar qualquer elemento pelo nome e obter, por exemplo, seu valor.

Veja o código a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var tag;
    </script>
  </head>
  <body>
    <form name="curso" action="#" method="post">
      <label>Nome</label><br>
      <input type="text" name="fNome" value="Digite seu nome"><br><br>
      <label>Senha</label><br>
      <input type="password" name="fSenha"><br><br>
      <input type="submit" name="btEnviar" value="Enviar">
    </form>
    <script>
      tag=document.forms["curso"];
      document.write(tag.elements["fNome"].value);
    </script>
  </body>
</html>
```

Este código gera o formulário e o script imprime o texto “Digite seu nome” que é o “value” do elemento “fNome”.

## Date - Trabalhando com data e hora

Em Javascript temos a nossa disposição uma classe com todos os métodos necessários para trabalhar com data e hora, é a classe “Date”, já está tudo pronto, basta chamar os métodos desejados, vamos aprender tudo sobre data e hora neste capítulo.

Uma observação importante é que a data e a hora retornadas pela classe Date, são a data e hora do computador do cliente e não do servidor, ou seja, o código busca a data e hora configurados em sua máquina, se seu relógio estiver errado a data mostrada será errada, portanto, cada pessoa poderá ver uma data e hora específicas que estiverem configuradas em seu computador.

Para trabalharmos com Data e Hora é simples, basta criar um objeto do tipo Date que já possui todos os métodos necessários para obtermos as informações da data e hora.

Vamos diretamente ao código para criar o objeto do tipo Date.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
```

```
<meta charset="UTF-8">
<script>
    var dataHora=new Date();
</script>
</head>
<body>

</body>
</html>
```

Viu como é simples? Desta maneira criamos um objeto do tipo Date com nome “dataHora”, porém ainda não obtivemos nenhum resultado visível.

Vamos usar os métodos para imprimir a data na tela.

```
<script>
    var data=new Date();
    var dia=data.getDate();
    var mes=data.getMonth()+1;
    var ano=data.getFullYear();
    document.write(dia + "/" + mes + "/" + ano + "<br>");
</script>
```

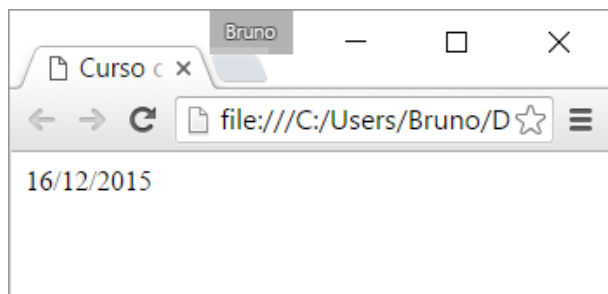
Vamos ver os detalhes dos métodos usados acima.

var dia=data.getDate(); → O método getDate() retorna o dia do mês, neste caso armazena na variável dia.

var mes=data.getMonth()+1; → O método getMonth() retorna o mês, mas em forma de vetor, então, como o primeiro elemento do vetor tem índice [0], janeiro = 0, fevereiro = 1, março = 2 e assim por diante, por isso adicionamos 1 ao final do método,.

var ano=data.getFullYear(); → O método getFullYear() retorna o ano com quatro dígitos.

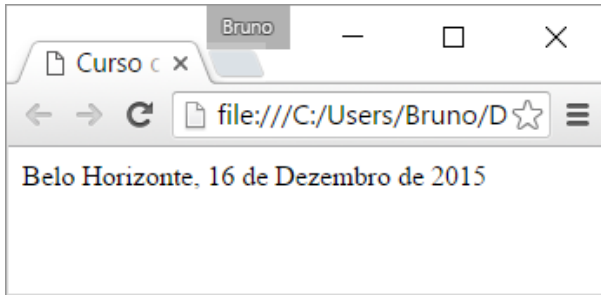
Então o resultado desta impressão é o mostrado na janela a seguir.



Vamos escrever a data por estêncil, altere o código para o mostrado a seguir.

```
<script>
    var data=new Date();
    var meses=new Array("Janeiro","Fevereiro","Março","Abril","Maio","Junho","Julho","Agosto","Setembro","Outubro","Novembro","Dezembro");
    var dia=data.getDate();
    var mes=data.getMonth();
    var ano=data.getFullYear();
    document.write("Belo Horizonte, " + dia + " de " + meses[mes] + " de " + ano + "<br>");
</script>
```

Criamos um array com os meses e usamos a variável “mes” como índice para este array, por último formatamos a impressão e o resultado é igual à ilustração a seguir.



Vou ver sobre os métodos mais importantes.

Método	Descrição	Exemplo
getDay();	Retorna o dia da semana, domingo=0, segunda=1	var diaSem=data.getDay();
getDate();	Retorna o dia do mês	var dia=data.getDate();
getMonth();	Retorna o mês, janeiro=0, fevereiro=1	var mes=data.getMonth();
getFullYear();	Retorna o ano com 4 dígitos, ex: 2015	var ano=data.getFullYear();
getHours();	Retorna a hora	var hora=data.getHours();
getMinutes();	Retorna os minutos	var minutos=data.getMinutes();
getSeconds();	Retorna os segundos	var segundos=data.getSeconds();
toString();	Retorna a data por estêncil, padrão EUA	var dataTexto=data.toString();
toLocaleDateString();	Retorna a data no formato 16/12/2015	var dataPadrao=data.toLocaleDateString();
toLocaleString();	Retorna a data e a hora 16/12/2015 23:13:00	var dataHoraTexto=data.toLocaleString();

Faça um teste com todos os métodos para ver o resultado.

## Math

Math é uma espécie de “biblioteca” disponível com vários métodos e constantes disponíveis para facilitar nosso trabalho com matemática, antes de aprendermos sobre como usar “math” vamos ver duas tabelas com as constantes e métodos disponíveis para o “math”.

Constante	Descrição
Math.E	Retorna o número Euler
Math.PI	Retorna o número PI
Math.SQRT2	Retorna a raiz quadrada de 2
Math.SQRT1_2	Retorna a raiz quadrada de ½
Math.LN2	Retorna o logaritmo natural de 2
Math.LN10	Retorna o logaritmo natural de 10
Math.LOG2E	Retorna base 2 do logaritmo de E
Math.LOG10E	Retorna base decimal do logaritmo de E

Método	Descrição
abs(x)	Retorna o valor absoluto de X
acos(x)	Retorna o arco cosseno de X
asin(x)	Retorna o arco seno de X
atan(x)	Retorna o arco tangente de X como um valor numérico entre PI/2 e PI/2 radiano
atan2(y,x)	Retorna o arco tangente do quociente dos argumentos y e x
ceil(x)	Retorna o valor de X arredondado para cima
cos(x)	Retorna o cosseno de x em radianos
exp(x)	Retorna o valor de Ex

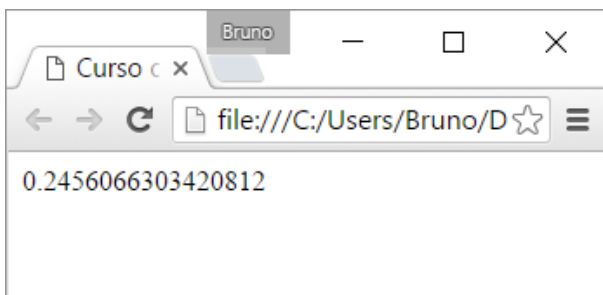
floor(x)	Retorna o valor de X arredondado para baixo
log(x)	Retorna o logaritmo natural (base E) de x
max(x,y,z,...,n)	Retorna o maior valor dos argumentos
min(x,y,z,...,n)	Retorna o menor valor dos argumentos
pow(x,y)	Retorna o valor de x elevado a y
random()	Retorna um número aleatório entre 0 e 1
round(x)	Retorna o inteiro mais próximo, arredonda para cima ou para baixo
sin(x)	Retorna o seno de x em radianos
sqrt(x)	Retorna a raiz quadrada de x
tan(x)	Retorna a tangente do ângulo x

Claro que não vou mostrar todos os métodos e as constantes, mas vou mostrar alguns a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var num=Math.random();
      document.write(num);
    </script>
  </head>
  <body>

  </body>
</html>
```

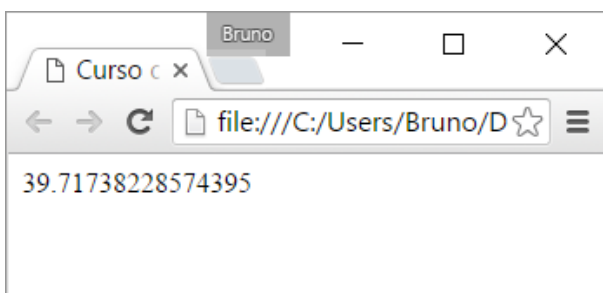
Este programa gera um número aleatório entre 0 e 1, esse número é gerado pelo método random() e armazenado na variável "num", em seguida este número é impresso na tela, confira o resultado com a ilustração a seguir, mas não compare exatamente o valor, pois, o método random gera um número aleatório diferente a cada chamada.



Podemos alterar para obter uma faixa diferente, suponhamos que precisemos de um número aleatório entre 0 e 100, então, basta multiplicar por 100 no final, veja a alteração.

```
<script>
  var num=Math.random()*100;
  document.write(num);
</script>
```

Veja o novo tipo de resultado.

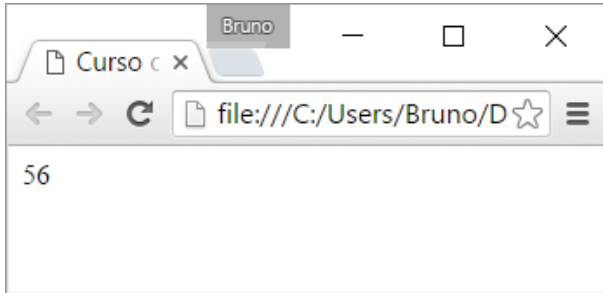




Vamos arredondar esse número, podemos usar uma das funções que fazem arredondamento ou retornam a parte inteira do número. A função floor por exemplo.

```
<script>
  var num=Math.floor(Math.random()*100);
  document.write(num);
</script>
```

No código acima estamos gerando o número entre 0 e 100 como parâmetro do método floor que retorna este número arredondado, então o tipo de resultado deste código é o mostrado a seguir.



Cada vez que atualizar a página “F5” será gerado um número diferente.

## Function

Uma ótima forma de controlar a execução de determinado bloco de código em um programa é utilizar funções, assim podemos criar toda uma rotina de programação e executar esta rotina em um momento específico, ter esse controle é extremamente útil e fundamental.

Quando criamos uma função, adicionamos uma série de comandos dentro desta função e estes comandos somente serão executados quando a função for chamada, ou seja, se em nenhum momento a função não for chamada, os comandos não serão executados, então, eu consigo determinar o momento exato que precisamos executar estes comandos que estão no escopo (corpo) da função.

Outro detalhe importante é quanto a economia de código, suponhamos que exista um bloco de comandos que precisa ser executado várias vezes em momentos diferentes em nosso programa, ao invés de repetir este bloco de comandos várias vezes, simplesmente adicione este bloco em uma função e sempre que precisar executar este bloco de comandos, basta chamar a função.

É bastante simples trabalhar com funções em Javascript, veja a seguir a sintaxe básica.

```
function nomeDaFunção(lista de argumentos){
  comandos;
  comandos;
  retorno da função;
}
```

Este é a sintaxe básica, mas podemos omitir alguns itens desta sintaxe, caso não sejam utilizados, como por exemplo, “lista de argumentos” ou “retorno da função”.

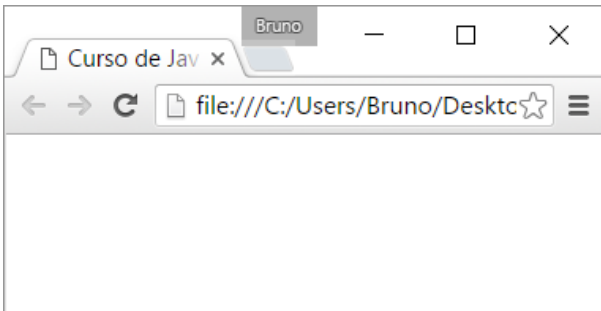
Vamos ver a seguir um programa que utiliza uma função bem simples.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      function escreve(){
```

```
        }  
        document.write("Canal Fessor Bruno<br>");  
    }  
</script>  
</head>  
<body>  
  
</body>  
</html>
```

No código acima criamos uma função chamada “escreve” com somente um comando “document.write” no seu escopo, então quando essa função for chamada será impresso na tela o texto “Canal Fessor Bruno”.

Se você rodar a página com esta função o resultado será uma página em branco.



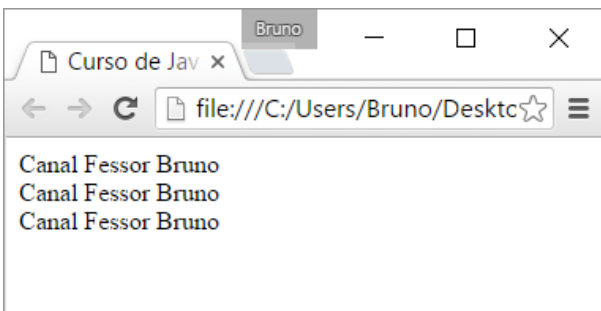
Calma, não deu nenhum erro, esse é o resultado esperado.

Não tem nada impresso na tela simplesmente porque a função não foi chamada, ele foi declarada mas não foi chamada.

Vamos alterar nosso código e adicionar algumas chamadas para a função.

```
<script>  
    function escreve(){  
        document.write("Canal Fessor Bruno<br>");  
    }  
  
    escreve();  
    escreve();  
    escreve();  
</script>
```

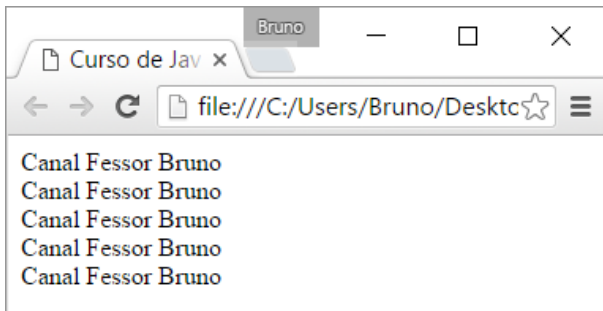
No código acima chamamos a função “escreve” três vezes e o resultado é semelhante à ilustração a seguir.



Podemos associar a função com um loop, por exemplo o loop FOR, para chamar a função um número determinado de vezes, confira o código a seguir.

```
<script>  
    function escreve(){  
        document.write("Canal Fessor Bruno<br>");  
    }  
  
    for(var i=0; i<5; i++){  
        escreve();  
    }  
</script>
```

Com este FOR chamamos a função “escreve” cinco vezes e o resultado é o da ilustração a seguir.



Vamos criar outro tipo de função, com argumentos de entrada, criaremos uma função que vai receber quatro notas de um aluno e imprimir se o aluno foi aprovado ou não.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      function notas(n1, n2, n3, n4){
        var soma=n1+n2+n3+n4;
        if(soma >= 60){
          document.write("<p style='color:#00F'>Aprovado</p>");
        }else{
          document.write("<p style='color:#F00'>Reprovado</p>");
        }
      }

      notas(20,10,15,5);
    </script>
  </head>
  <body>

  </body>
</html>
```

Na função anterior “notas” podemos perceber que configuramos quatro parâmetros de entrada (n1, n2, n3, n4), a utilização de parâmetros em funções têm o objetivo de permitir a entrada de valores para dentro da função, esses valores são passados de “fora” para “dentro” da função no momento da chamada.

Para realizar a chamada deste função precisamos informar os valores correspondentes para cada um dos parâmetros, então a chamada notas(20,10,15,5); adicione para o parâmetro n1 o valor 20, para n2 o valor 10, para o parâmetro n3 o valor 15 e para o parâmetro n4 o valor 5.

Já dentro da função somamos estes parâmetros e armazenamos seu valor na variável soma.

```
var soma=n1+n2+n3+n4;
```

Na sequência usamos o IF para verificar a nota e imprimir “Aprovado” ou “Reprovado”.

Vamos aumentar um pouco a dificuldade do nosso programa, vamos criar uma matriz, array bidimensional, adicionar 4 notas para 8 alunos e verificar os alunos que foram aprovados ou reprovados.

```
<script>
  var alunos=new Array();
  var aluno=new Array();
  var i;

  function notas(n1, n2, n3, n4){
    var soma=n1+n2+n3+n4;
    if(soma >= 60){
      document.write("<span style='color:#00F'>Aprovado</span><br>");
    }else{
      document.write("<span style='color:#F00'>Reprovado</span><br>");
    }
  }
```

```

    }

    aluno=["Godofreudo",20,10,15,5];
    alunos[0]=aluno;
    aluno=["Filisberta",20,20,10,30];
    alunos[1]=aluno;
    aluno=["Craudiomiro",5,5,7,2];
    alunos[2]=aluno;
    aluno=["Nevernilda",9,15,15,20];
    alunos[3]=aluno;
    aluno=["Jubisberto",20,20,30,30];
    alunos[4]=aluno;
    aluno=["Franclara",5,15,10,20];
    alunos[5]=aluno;
    aluno=["Admirson",15,15,25,25];
    alunos[6]=aluno;
    aluno=["Romicreide",20,19,30,25];
    alunos[7]=aluno;

    for(i=0; i<alunos.length; i++){
        document.write("Aluno: " + alunos[i][0] + " = ");
        notas(alunos[i][1], alunos[i][2], alunos[i][3], alunos[i][4]);
    }
}
</script>

```

Vou detalhar passo a passo o código acima.

Primeiramente declaramos nossos arrays e a variável para servir de índice no FOR que irá percorrer nosso array alunos.

```

var alunos=new Array();
var aluno=new Array();
var i;

```

Na sequência criamos a função que irá imprimir se o aluno foi aprovado ou reprovado, é a mesma função que usamos anteriormente com uma pequena alteração na impressão, ao invés de usar a tag <p> trocamos para tag <span>.

```

function notas(n1, n2, n3, n4){
    var soma=n1+n2+n3+n4;
    if(soma >= 60){
        document.write("<span style='color:#00F'>Aprovado</span><br>");
    }else{
        document.write("<span style='color:#F00'>Reprovado</span><br>");
    }
}

```

Os próximos comandos são para preenchimento do array alunos com as informações dos alunos, nome e quatro notas.

```

aluno=["Godofreudo",20,10,15,5];
alunos[0]=aluno;

```

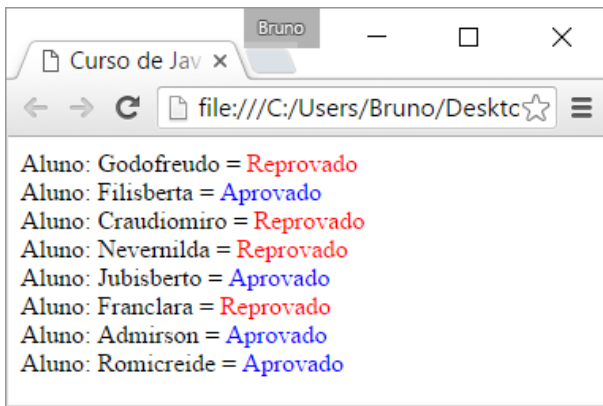
Por último usamos um loop FOR que imprime o nome do aluno na posição alunos[i][0], e passamos as notas do aluno "i" para a função notas, que faz a devida verificação e imprime "Aprovado" ou "Reprovado".

```

for(i=0; i<alunos.length; i++){
    document.write("Aluno: "+alunos[i][0]+ " = ");
    notas(alunos[i][1], alunos[i][2], alunos[i][3], alunos[i][4]);
}

```

Veja a seguir o resultado deste código.



Outro ponto importante que precisamos discutir sobre funções é o retorno, podemos criar funções que retornam valores se saída para quem vez a chamada da função, assim, podemos associar uma função com uma variável que irá armazenar este valor de retorno.

Vamos alterar nossa função notas, para retornar o valor da nota, veja a seguir as alterações no código, destacadas em vermelho.

```
<script>
    var alunos=new Array();
    var aluno=new Array();
    var i,nota;

    function notas(n1, n2, n3, n4){
        var soma=n1+n2+n3+n4;
        if(soma >= 60){
            document.write("<span style='color:#00F'>Aprovado</span>");
        }else{
            document.write("<span style='color:#F00'>Reprovado</span>");
        }
        return soma;
    }

    aluno=["Godofreudo",20,10,15,5];
    alunos[0]=aluno;
    aluno=["Filisberta",20,20,10,30];
    alunos[1]=aluno;
    aluno=["Craudimiro",5,5,7,2];
    alunos[2]=aluno;
    aluno=["Nevernilda",9,15,15,20];
    alunos[3]=aluno;
    aluno=["Jubisberto",20,20,30,30];
    alunos[4]=aluno;
    aluno=["Franclara",5,15,10,20];
    alunos[5]=aluno;
    aluno=["Admirson",15,15,25,25];
    alunos[6]=aluno;
    aluno=["Romicreide",20,19,30,25];
    alunos[7]=aluno;

    for(i=0; i<alunos.length; i++){
        document.write("Aluno: "+alunos[i][0]+ " = ");
        nota=notas(alunos[i][1], alunos[i][2], alunos[i][3], alunos[i][4]);
        document.write(" - Nota: " + nota + "<br>");
    }
</script>
```

Note que agora nossa função retorna o valor da soma no final, ou seja, funcionará da mesma forma que anteriormente, porém, ao final retorna o valor da soma.

```
function notas(n1, n2, n3, n4){
    var soma=n1+n2+n3+n4;
    if(soma >= 60){
```

```

        document.write("<span style='color:#00F'>Aprovado</span>");
    }else{
        document.write("<span style='color:#F00'>Reprovado</span>");
    }
    return soma;
}

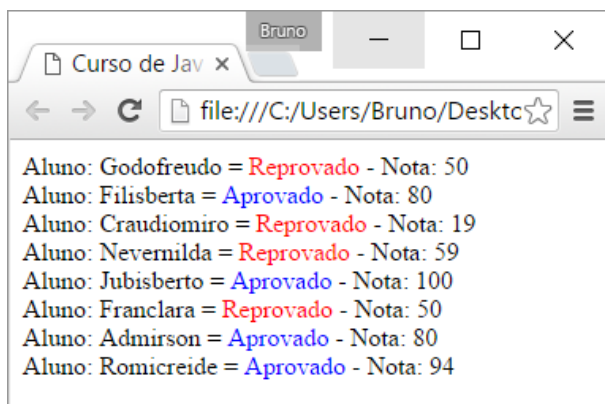
```

Neste caso a chamada da função será um pouco diferente, como nossa função retorna um valor, precisamos associar o retorno da função a uma variável, no caso, a variável `notas`, observe a chamada da função.

```
nota=notas(alunos[i][1], alunos[i][2], alunos[i][3], alunos[i][4]);
```

A variável `notas` armazena a soma das notas que foi feita dentro da função e retornada no final.

Desta forma podemos imprimir o valor da variável `nota` que conterà a soma das notas de cada um dos alunos, observe o resultado a seguir.



Vamos criar um novo programa com uma função que irá receber uma série de números através de um array, calcular e retornar a média destes números.

```

<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var numeros=new Array();
      var resultado;

      function media(nums){
        var tam=nums.length;
        var soma=0;
        for(i=0; i< tam; i++){
          soma+=nums[i];
        }

        return Math.floor(soma/tam);
      }

      resultado=media([10,2,5,30,25,19]);

      document.write("Média: " + resultado);
    </script>
  </head>
  <body>

  </body>
</html>

```

Vou detalhar a função.

`function media(nums){` → Criamos a função com um parâmetro de entrada que irá receber um array.

`var tam=nums.length;` → Obtivemos o tamanho do array (quantidade de números) passado e armazenamos na variável `tam`.

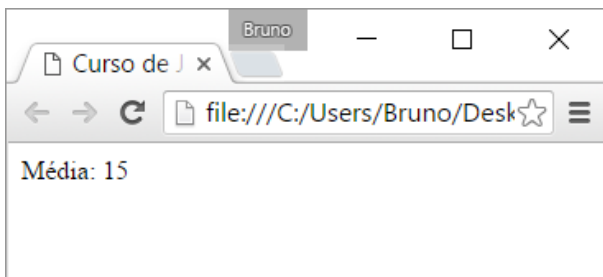
`var soma=0;` → Criamos e iniciamos a variável `soma` com valor zero, conterá a soma dos números do array.

Loop FOR para percorrer o array e somar todos os elementos guardando a soma na variável `soma`.

```
for(i=0; i< tam; i++){  
    soma+=nums[i];  
}
```

`return Math.floor(soma/tam);` → Retorna o valor da média arredondado.

O resultado deste código pode ser visto a seguir.



Agora que já sabemos sobre funções as possibilidades aumentam ainda mais e a organização dos nossos códigos também irá aumentar.

## *Escopo, variáveis locais e globais*

Basicamente escopo é o limite de alcance que um determinado elemento tem, ou até por onde um determinado elemento ou variável pode ser alcançada, por exemplo, se criar uma variável local dentro de uma função esta variável só poderá ser utilizada por esta função, outras funções ou comandos fora da função inicial não poderão chegar até esta variável.

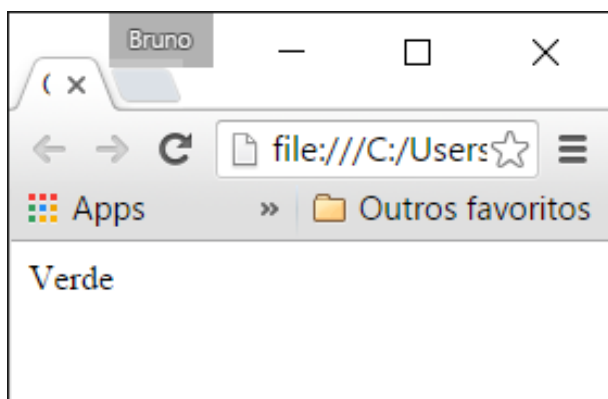
Por isso precisamos entender o conceito de variáveis locais e globais em javascript.

É importante entender de escopo para que não tenhamos problemas na hora de saber se uma variável que tem seu valor alterado dentro de uma função terá o mesmo valor fora da função, por exemplo.

Veja um código simples de exemplo.

```
<!doctype html>  
<html lang="pt-br">  
  <head>  
    <title>Curso de Javascript</title>  
    <meta charset="UTF-8">  
    <script>  
  
      var cor="Verde";  
  
      function escreveCor(){  
        document.write(cor);  
      }  
  
      escreveCor();  
  
    </script>  
  </head>  
  <body>  
  
  </body>  
</html>
```

O resultado deste código é a impressão na tela do texto “Verde”.



Simple e óbvio! A variável “cor” que está sendo usada dentro da função é a mesma variável que está sendo declarada e atribuída fora da função, isso porque ela está em um contexto de variável global, toda variável criada diretamente na área de script, fora de uma função, é uma variável global, ou seja, pode ser acessada de qualquer lugar do código.

Agora veja uma pequena alteração do programa que não afetará seu funcionamento, mas é importante para entendermos o conceito de local e global.

```
<script>

    var cor="Verde";

    function escreveCor(){
        cor;
        document.write(cor);
    }

    escreveCor();

</script>
```

Como disse esta alteração não irá afetar o resultado do programa, pois, simplesmente informamos o nome da variável dentro da função, como não atribuímos um novo valor, continua imprimindo o texto “Verde”.

Agora vamos a alteração importante, adicione a palavra reservada “var” antes do nome da variável.

```
<script>

    var cor="Verde";

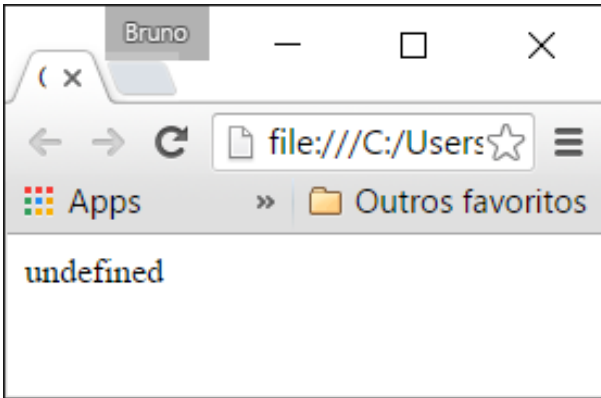
    function escreveCor(){
        var cor;
        document.write(cor);
    }

    escreveCor();

</script>
```

Agora veja o resultado desta alteração.





O que aconteceu? Porque o valor da variável está como não definido? É um erro de programação?

O que aconteceu foi simplesmente que ao adicionar a palavra reservada “var” antes de uma variável, estamos definindo-a como local, ou seja, só pode ser acessada do local onde foi criada por elementos que estejam no mesmo escopo.

Então neste caso temos duas variáveis com nome “cor” uma fora da função disponível para todo script e outra dentro da função, disponível somente no escopo da função.

Por isso o resultado está como não definida, por que ela não possui valor nenhuma atribuído, a variável que tem uma valor atribuído é a que está fora da função.

Vamos atribuir um valor para a variável “cor” que está dentro da função e ver o resultado.

```
<script>

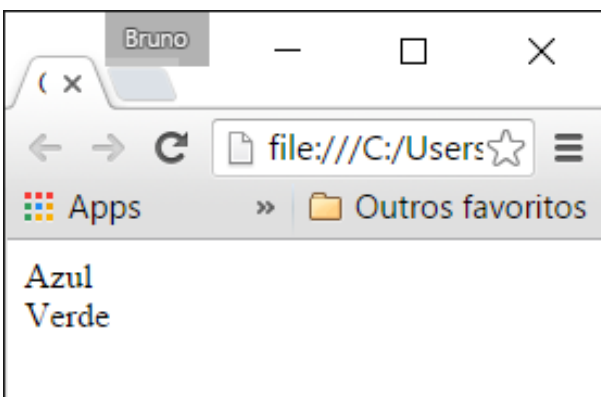
  var cor="Verde";

  function escreveCor(){
    var cor="Azul";
    document.write(cor);
  }

  escreveCor();

  document.write("<br>" + cor);

</script>
```



Note que foram impressos os textos “Azul” e “Verde” referentes a variável “cor” dentro da função escreveCor() e a variável “cor” de fora da função.

Se tirar a palavra reservada “var” de dentro da função veja o resultado.

```
function escreveCor(){
  cor="Azul";
  document.write(cor);
}
```

OBS: Caso a variável seja criada na área de código, fora de uma função, tanto faz, usar a palavra “var” ou não.

## Eventos

Eventos são acontecimentos específicos em nossa página, toda ação do usuário em uma página gera um evento, por exemplo, quando movemos o mouse, quando clicamos com o botão do mouse, quando a página é carregada, quando selecionamos um item, etc.

Os eventos são disparados por determinados itens, um item pode disparar mais de um tipo de evento, por exemplo, vamos imaginar o simples ato de posicionar o mouse sobre um botão submit de formulário, clicar e retirar o mouse de cima do botão, consegue contar quantos e quais os eventos disparados nesta ação? Vamos conferir os eventos disparados pelo botão submit.

Estado inicial, não é disparado nenhum evento.



Ao mover o mouse e posicionar o cursor sobre o botão, são disparados dois eventos:

onmouseover: Quando o cursor fica sobre o botão.

onmousemove: Toda vez que o cursor se mover sobre o botão.



No momento do clique quando pressionamos o botão para baixo são disparados os eventos:

onclick: Evento padrão de clique em elementos.

onmousedown: Evento disparado quando o botão do mouse está para baixo, ou seja, quando é clicado.



No momento em que o botão do mouse é solto, terminando o clique é disparado o evento onmouseup.



Ao mover o cursor para fora do botão são disparados os eventos:

onmouseout: Toda vez que o cursor se mover sobre o botão.

onmouseout: Evento disparado quando o mouse sai de cima do botão.



Nossa tarefa é descobrir qual evento precisamos de interceptar e adicionar o código para este evento, desta maneira, o código só será executado quando o evento for disparado.

Por exemplo, se quiser executar uma determinada função sempre que um botão for clicado, então deveremos adicionar a chamada desta função no evento onclick deste botão.

Confira a seguir as tabelas com a listagem completa dos eventos.

### Eventos de mouse

Evento	Descrição
onclick	Evento disparado quando se clica em um elemento.
oncontextmenu	Evento disparado quando se clica com o botão direito do mouse sobre um elemento.
ondblclick	Evento disparado quando é aplicado um clique duplo sobre um elemento.
onmousedown	Evento disparado quando o botão do mouse é pressionado sobre um elemento.
onmouseenter	Evento disparado quando o ponteiro do mouse se move para cima de um elemento.
onmouseleave	Evento disparado quando o ponteiro do mouse se move para fora de um elemento.
onmousemove	Evento disparado quando o ponteiro do mouse se move sobre um elemento.
onmouseover	Evento disparado quando o ponteiro do mouse é movido para dentro de um elemento ou um de seus filhos.
onmouseout	Evento disparado quando o ponteiro do mouse é movido para fora de um elemento ou um de seus filhos.
onmouseup	Evento disparado quando o botão do mouse é liberado, despreciando, sobre um elemento.

### Eventos de teclado

Evento	Descrição
onkeydown	Evento disparado quando o usuário pressiona uma tecla.
onkeypress	Evento disparado quando o usuário mantém uma tecla pressionada.
onkeyup	Evento disparado quando o usuário libera uma tecla pressionada.

### Eventos de objetos/frames/body

Evento	Descrição
onabort	Evento disparado quando o carregamento do elemento é abortado.
onbeforeunload	Evento disparado antes do documento ser descarregado/fechado.
onerror	Evento disparado quando ocorre algum erro no carregamento de arquivos externos.
onhashchange	Evento disparado quando ocorre alguma alteração da âncora da URL.
onload	Evento disparado quando um elemento é carregado.
onpageshow	Evento disparado quando uma página é mostrada.
onpagehide	Evento disparado quando uma página deixa de ser mostrada.
onresize	Evento disparado quando o elemento é redimensionado.
onscroll	Evento disparado quando a página é rolada.
onunload	Evento disparado quando a página é fechada/d Descarregada.

## Eventos de formulários

Evento	Descrição
onblur	Evento disparado quando o elemento perde o foco, perde a seleção.
onchange	Evento disparado quando o conteúdo do elemento é alterado, input, keygen, select, textarea.
onfocus	Evento disparado quando o elemento recebe o foco, é selecionado.
onfocusin	Evento disparado quando o elemento está prestes a receber o foco.
onfocusout	Evento disparado quando o elemento está prestes a perder o foco.
oninput	Evento disparado quando o usuário clica para entrar ou selecionar o elemento.
oninvalid	Evento disparado quando um elemento é inválido.
onreset	Evento disparado quando o botão reset é pressionado.
onsearch	Evento disparado quando é escrito algo em um campo <input type="search">
onselect	Evento disparado logo após a seleção de um texto <text> e <textarea>
onsubmit	Evento disparado quando o botão submit é clicado para enviar os dados do formulário.

## Eventos de Drag / Arrastar

Evento	Descrição
ondrag	Evento disparado quando um elemento é arrastado.
ondragend	Evento disparado quando um elemento deixa de ser arrastado.
ondragenter	Evento disparado quando o elemento arrastado entra no elemento alvo.
ondragleave	Evento disparado quando o elemento arrastado deixa o elemento alvo.
ondragover	Evento disparado quando o elemento arrastado está sobre o elemento alvo.
ondragstart	Evento disparado quando o elemento começa a ser arrastado.
ondrop	Evento disparado quando o elemento arrastado é solto.

## Eventos de Área de transferência / clipboard

Evento	Descrição
oncopy	Evento disparado quando o usuário copia o conteúdo de um elemento.
oncut	Evento disparado quando o usuário recorta o conteúdo de um elemento.
onpaste	Evento disparado quando o usuário cola um conteúdo em um elemento.

## Eventos de impressão

Evento	Descrição
onafterprint	Evento disparado quando a página termina de ser impressa.
onbeforeprint	Evento disparado quando a página está prestes a ser impressa.

## Eventos de mídia

Evento	Descrição
onabort	Evento disparado quando o carregamento de uma mídia é interrompido.
oncanplay	Evento disparado quando a página está prestes a ser impressa.
oncanplaythrough	Evento disparado quando o navegador pode começar a rodar a mídia (quando se tem cach suficiente para começar)
ondurationchange	Evento disparado quando a duração da mídia é alterada.

onemptied	Evento disparado quando algum problema acontece e o arquivo de mídia fica subitamente indisponível (quando a conexão é inesperadamente interrompida)
onended	Evento disparado quando a mídia chegou ao fim (útil para mensagens como "Obrigado por escutar")
onerror	Evento disparado quando ocorreu um erro durante o carregamento de um arquivo de mídia.
onloadeddata	Evento disparado quando os dados de mídia são carregados.
onloadedmetadata	Evento disparado quando dados de meta (como tamanho e duração) são carregados
onloadstart	Evento disparado quando o navegador começa a carregar a mídia especificada
onpause	Evento disparado quando a mídia é pausada pelo usuário.
onplay	Evento disparado quando a mídia é iniciada ou não está pausada.
onplaying	Evento disparado quando a mídia está sendo tocada.
onprogress	Evento disparado quando o navegador baixando a mídia.
onratechange	Evento disparado quando a velocidade de reprodução da mídia é alterada.
onseeked	Evento disparado quando o usuário termina de mover o ponteiro de reprodução para uma nova posição da mídia.
onseeking	Evento disparado quando o usuário começa a mover o ponteiro de reprodução para uma nova posição da mídia.
onstalled	Evento disparado quando o navegador está tentando obter dados de mídia, mas os dados não estão disponíveis.
onsuspend	Evento disparado quando a reprodução da mídia é suspenso, quando o navegador não está recebendo dados da mídia.
ontimeupdate	Evento disparado quando a posição de reprodução mudar (como quando o usuário avança para um ponto diferente na mídia)
onvolumechange	Evento disparado quando o volume é alterado.
onwaiting	Evento disparado quando a mídia fez uma pausa, mas é esperado para retomar (quando a mídia faz uma pausa para o bufferizar mais dados)

### Eventos de animação

Evento	Descrição
animationend	Evento disparado quando uma animação CSS é completada.
animationiteration	Evento disparado quando uma animação CSS é repetida.
animationstart	Evento disparado quando uma animação CSS é iniciada.

### Evento de transição

Evento	Descrição
transitionend	Evento disparado quando uma transição CSS é completada.

### Eventos enviados pelo servidor

Evento	Descrição
onerror	Evento disparado quando um erro ocorre.
onmessage	Evento disparado quando uma mensagem é recebida através da fonte de eventos.
onopen	Evento disparado quando a conexão com o servidor é iniciada.

### Eventos diversos

Evento	Descrição
--------	-----------

onmessage	Evento disparado quando uma mensagem é recebida através ou a partir de um objeto.
ononline	Evento disparado quando o browser inicia o trabalho online.
onoffline	Evento disparado quando o browser inicia o trabalho offline.
onpopstate	Evento disparado quando a janela de histórico muda.
onshow	Evento disparado quando um elemento de <menu> é mostrado.
onstorage	Evento disparado quando a área de armazenamento web é atualizada.
ontoggle	Evento disparado quando o usuário abre ou fecha um elemento de <details>
onwheel	Evento disparado quando a roda do mouse é usada.

## Eventos de toque na tela

Evento	Descrição
ontouchcancel	Evento disparado quando o toque é finalizado.
ontouchend	Evento disparado quando o dedo é retirado na tela de toque.
ontouchmove	Evento disparado quando o dedo é movido na tela de toque.
ontouchstart	Evento disparado quando o dedo toca a tela.

## Event

Evento	Descrição
DOMContentLoaded	Acionado quando o documento inicial HTML foi completamente carregado e analisado.

## MouseEvent

Evento	Descrição
altKey	Retorna se a tecla ALT foi pressionada.
button	Retorna qual botão do mouse foi pressionado.
buttons	Retorna quais botões do mouse foram pressionados.
clientX	Retorna a coordenada horizontal do mouse.
clientY	Retorna a coordenada vertical do mouse.
ctrlKey	Retorna se a tecla CTRL foi pressionada.
detail	Retorna um número que indica quantas vezes o mouse foi clicado.
metaKey	Retorna se a tecla "META" foi pressionada.
relatedTarget	Retorna o elemento relacionado com o elemento que disparou o evento do mouse.
screenX	Retorna a coordenada horizontal do ponteiro do mouse, em relação à tela.
screenY	Retorna a coordenada vertical do ponteiro do mouse, em relação à tela.
shiftKey	Retorna se a tecla SHIFT foi pressionada.
which	Retorna qual botão do mouse foi pressionado.

## KeyboardEvent

Evento	Descrição
altKey	Retorna se a tecla ALT foi pressionada.
ctrlKey	Retorna se a tecla CTRL foi pressionada.
charCode	Retorna o código do caracteres da tecla pressionada.
key	Retorna o valor da tecla pressionada.
keyCode	Retorna o código da tecla pressionada.
location	Retorna a localização da tecla pressionada.
metaKey	Retorna se a tecla "META" foi pressionada.

shiftKey	Retorna se a tecla SHIFT foi pressionada.
which	Retorna qual botão do mouse foi pressionado.

### HashChangeEvent

Evento	Descrição
newURL	Retorna a URL do documento após uma alteração.
oldURL	Retorna a URL do documento antes uma alteração.

### PageTransitionEvent

Evento	Descrição
persisted	Retorna se a página da Web foi armazenada em cache pelo navegador.

### FocusEvent

Evento	Descrição
relatedTarget	Retorna o elemento relacionado com o elemento que disparou o evento.

### AnimationEvent

Evento	Descrição
animationName	Retorna o nome da animação.
elapsedTime	Retorna o tempo em segundos que uma animação está rodando.

### TransitionEvent

Evento	Descrição
propertyName	Retorna o nome da propriedade CSS associada com a transição.
elapsedTime	Retorna o tempo em segundos que uma transição está rodando.

### WheelEvent

Evento	Descrição
deltaX	Retorna a quantidade de rolagem horizontal de uma roda do mouse (eixo x).
deltaY	Retorna a quantidade de rolagem vertical de uma roda do mouse (eixo y).
deltaZ	Retorna a quantidade de deslocamento de uma roda do mouse para o eixo z.
deltaMode	Retorna um número que representa a unidade de medição para valores delta (pixels, linhas ou páginas).

A variedade de eventos é muito grande, podemos interceptar praticamente qualquer acontecimento no computador em relação à ao browser.

Vamos ver alguns exemplos de utilização dos eventos, os eventos estão destacados em vermelho.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
```

```

<meta charset="UTF-8">
<script>
    function cumprimento() {
        var data=new Date();
        hora=data.getHours();
        if((hora >= 6)&&(hora <= 12)){
            alert("Bom dia");
        }else if((hora > 12)&&(hora < 18)){
            alert("Boa tarde");
        }else if(hora >= 18){
            alert("Boa noite");
        }else{
            alert("Boa madrugada");
        }
    }

    function msg1(){
        alert("Você clicou no botão");
    }

    function msg2(){
        alert("Cursos passou sobre a imagem");
    }
</script>
</head>
<body onload="cumprimento()">

    <input type="button" value="Clique-me" onclick="msg1()">
    <br>
    

</body>
</html>

```

Vamos detalhar as partes mais importantes deste script.

Primeiramente criamos uma função que verifica a hora e de acordo com este valor emite uma caixa de mensagem com o cumprimento adequado.

```

function cumprimento(){
    var data=new Date();
    hora=data.getHours();
    if((hora >= 6)&&(hora <= 12)){
        alert("Bom dia");
    }else if((hora > 12)&&(hora < 18)){
        alert("Boa tarde");
    }else if(hora >= 18){
        alert("Boa noite");
    }else{
        alert("Boa madrugada");
    }
}

```

Na sequência criamos outras duas funções simplesmente para emitir mensagens em caixas do tipo alert.

```

function msg1(){
    alert("Você clicou no botão");
}

function msg2(){
    alert("Cursor passou sobre a imagem");
}

```

Com as funções prontas vamos ver os eventos associados.



`<body onload="cumprimento()">` → No evento onload do `<body>` associamos a função “cumprimento()”, ou seja, quando a página for aberta/carregada será emitida uma caixa alert com o cumprimento adequado ao horário.

`<input type="button" value="Clique-me" onclick="msg1()">` → Neste input estamos monitorando o evento onclick, ou seja, quando clicar no botão será mostrada a caixa alert com o texto "Você clicou no botão".

`` → Nesta imagem estamos associando a função “msg2()” com o evento “onmouseover”, ou seja, toda vez que posicionarmos o curso do mouse sobre a imagem será mostrada a mensagem “Cursos passou sobre a imagem”.

OBS: A tag `<img>` usada no código anterior não mostrará nenhuma imagem, pois, não utilizamos imagem no parâmetro src, propositalmente, será mostrado então o texto definido no parâmetro “alt”.

Podemos trabalhar com adição de listeners/escutadores para os eventos nos elementos, esse será o próximo assunto.

## *addEventListener*

Outra maneira de adicionar eventos aos elementos html é usando o comando `addEventListener`, uma das vantagens de se usar este método para adicionar eventos é que não dependemos do HTML, assim podemos deixar todo código javascript em um arquivo separado.

Vamos ver a seguir um código mostrando o uso do método `addEventListener`.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

      function msg1(){
        alert("Seja bem vindo");
      }

      window.addEventListener("load",msg1);

    </script>
  </head>
  <body>

  </body>
</html>
```

No exemplo acima, adicionamos um “listener/escutador” para o evento “load” ao window, neste caso quando a página for carregada a função “msg1()” será executada e a caixa de mensagem alert será mostrada.

OBS: Quando o evento for adicionado ao window, não precisamos informar “window”, pode ser omitido, os exemplos a seguir são semelhantes.

```
window.addEventListener("load",load);
addEventListener("load",load);
```

Vamos ver outro código onde adicionamos o evento “click” a um botão, para esta tarefa veremos uma particularidade, veja o código a seguir, parece muito lógico, mas este código não irá dar resultado nenhum.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
```

```
function msg1(){
    alert("Seja bem vindo");
}

var b1=document.getElementById("bt1");
b1.addEventListener("click",msg1);

</script>
</head>
<body>

    <input type="button" value="Clique-me" id="bt1">

</body>
</html>
```

No comando acima temos a função “msg1()” bem definida, sem nenhum problema.

Capturamos o elemento de id “bt1” e associamos à variável “b1”.

Adicionamos o listener para o evento “click” ao elemento “bt1” pela variável “b1” usando a função “msg1()” como ação para este evento.

Mas se tudo está correto, porque não funciona?

O problema aqui é bem simples, mas que faz muitas pessoas quebrarem a cabeça. Simplesmente porque o elemento “button” não foi carregado no momento em que adicionamos o evento, então, o evento não conseguiu achar seu “alvo”.

É um problema de carregamento, como o addEventListener está sendo executado primeiro e só depois o botão está sendo renderizado na página o listener não consegue achar o “bt1”, porque ele ainda não existe.

Como resolver este problema?

A resolução deste problema também é simples, basta adicionar um evento para o load do window e neste evento carregar uma função que adiciona todos os outros eventos, veja a alteração do código a seguir.

```
<script>

function load(){
    var b1=document.getElementById("bt1");
    b1.addEventListener("click",msg1);
}

function msg1(){
    alert("Seja bem vindo");
}

window.addEventListener("load",load);

</script>
```

Veja que agora criamos uma função com o nome “load()” e dentro desta função capturamos o elemento “bt1” e adicionamos o evento “click” a ele.

No final adicionamos o evento “load” no window e adicionamos a função “load()” como ação deste evento, ou seja, somente quando a página for carregada será adicionado o evento ao botão.

Agora Quando clicar no botão será mostrada a mensagem “Seja bem vindo”.

**OBS: Os eventos que podemos adicionar com o método addEventListener são os mesmos descritos nas tabelas anteriores com uma pequena diferença, não usamos o “on”, então, ao invés de usar “onclick” usamos apenas “click”.**

Vamos conhecer um pouco mais sobre addEventListener.

Vamos criar um código programa que captura as teclas direcionais do teclado e move um quadrado na tela de acordo com a tecla pressionada.

Primeiramente vamos criar o quadrado, será uma <div> formatada como um quadrado preto.

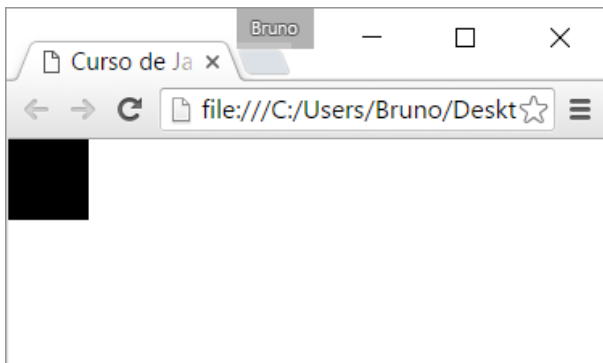
```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <style>
      #jogador{
        position:absolute;
        left:0px;
        top:0px;
        width:50px;
        height:50px;
        background-color:#000;
      }
    </style>
    <script>

  </script>
</head>
<body>

  <div id="jogador"></div>

</body>
</html>
```

Este é o resultado do código acima.



Agora vamos ao script.

```
<script>
  var px=0;
  var py=0;

  function load(){
    document.addEventListener("keydown", tecla);
  }

  function tecla(){
    var jg=document.getElementById("jogador");
    var tecla = event.keyCode;
    if (tecla==37){
      //Esquerda
      px-=10;
      jg.style.left=px+"px";
    }else if (tecla==39){
      //Direita
      px+=10;
      jg.style.left=px+"px";
    }
    if (tecla==38){
      //Cima
      py-=10;
      jg.style.top=py+"px";
    }
  }
}
```

```

        }else if(tecla==40){
            //baixo
            py+=10;
            jg.style.top=py+"px";
        }
    }

    window.addEventListener("load", load);
</script>

```

Vamos aos detalhes do código acima.

Primeiramente criamos duas variáveis px e py, para controlar a posição X e posição Y do quadrado na tela.

```
var px=0;
```

```
var py=0;
```

Em seguida criamos uma função com nome “load()” que será associada ao evento “load” do window, neste função programamos um listener para o evento “keydown” onde teremos uma função que irá verificar qual tecla foi pressionada, no final do código vamos chamar a função “load()” no evento “load” do window.

```
function load(){
    document.addEventListener("keydown",tecla);
}

```

Na sequência temos a função “tecla” que irá associar a <div id=“jogador”> à variável “jg”, criamos também a variável “vtecla” que irá receber o código da tecla pressionada, retornada pelo evento “keydown”.

```
function tecla(){
    var jg=document.getElementById("jogador");
    var vtecla = event.keyCode;
}

```

Ainda dentro da função programamos as estruturas IF que verificam o código da tecla pressionada, incrementam ou decrementam as variáveis px ou py de acordo com a tecla e atualiza a posição da <div id=“jogador”> de acordo com o valor das variáveis px ou py.

```

if(vtecla==37){
    //Esquerda
    px-=10;
    jg.style.left=px+"px";
}else if(vtecla==39){
    //Direita
    px+=10;
    jg.style.left=px+"px";
}
if(vtecla==38){
    //Cima
    py-=10;
    jg.style.top=py+"px";
}else if(vtecla==40){
    //baixo
    py+=10;
    jg.style.top=py+"px";
}

```

Os códigos das teclas são:

37 = Tecla direcional seta esquerda;

38 = Tecla direcional seta para cima;

39 = Tecla direcional seta direita;

40 = Tecla direcional seta para baixo.

Por último adicionamos o evento “load” ao window que chamará a função “load()” assim que a página for carregada.

```
window.addEventListener("load", load);
```

Salve as alterações e atualize a página para ver o código funcionar, veja que o quadrado se move de acordo com as teclas direcionais pressionadas.

## *removeEventListener()*

Da mesma forma que podemos adicionar eventos, também podemos remover os eventos, vamos criar um programa que captura a tecla pressionada e mostra o código desta tecla em uma caixa alert, quando a tecla <<ENTER>> for pressionada, código 13, o evento será removido.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      function load() {
        document.addEventListener("keydown", tecla);
      }

      function tecla() {
        var vtecla = event.keyCode;
        if(vtecla==13) {
          alert("Tecla <<ENTER>> pressionada. \n\n O evento será removido. \n\n");
          document.removeEventListener("keydown", tecla);
        } else {
          alert("Tecla pressionada: "+vtecla);
        }
      }

      window.addEventListener("load", load);
    </script>
  </head>
  <body>

  </body>
</html>
```

Salve e faça o teste do código, pressione algumas teclas e depois a tecla <<ENTER>>, note que após pressionar <<ENTER>> não será mostrado mais nenhum código de tecla, pois, o evento “keydown” foi removido.

## *Eventos de controle de tempo/intervalo (cronometragem) / timing events*

Estão disponíveis em javascript duas funções para controle de tempo, funções temporizadores, embora tenham a sintaxe bem parecida possuem uma pequena, porém, importante diferença em seu funcionamento.

São as funções “setTimeout” e “setInterval”.

Mas para que serve estas funções?

Basicamente funções “temporizadoras” chamam outras funções, callback, em um tempo determinado, podendo repetir esta chamada sempre ou não.

Tente imagina um relógio que a cada segundo deve atualizar a informação da hora mostrada, então a cada segundo deve ser chamada uma função que calcule e atualize a hora mostrada.

Vamos pensar em outra situação, um script de slider que mostra um novo slide de X em X segundos, então precisamos de um função de controle de tempo que faça esta tarefa.

Vamos aprender como usar estas funções.

## setTimeout

Como já foi dito anteriormente esse tipo de função serve para chamar outras funções (callback) em um tempo X determinado, é exatamente isso que setTimeout faz, basta configurar qual função vai ser chamada e o tempo, uma característica de setTimeout é que realiza a chamada somente uma vez, ou seja, ela não repete a chamada da função callback.

A sintaxe é bem simples, confira a seguir.

setTimeout(função callback, tempo);

**OBS: O parâmetro “tempo” é informado em milissegundos, ou seja, para configurar 5 segundos usamos o valor 5000, para configurar 1 segundo usamos os valor 1000.**

Vamos ver um script simples de como usar setTimeout.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      var timer=setTimeout(cfb,5000);

      function cfb(){
        document.write("Canal Fessor Bruno");
      }
    </script>
  </head>
  <body>

  </body>
</html>
```

No script acima configuramos um intervalo “setTimeout” que irá chamar a função “cfb” depois de cinco segundos, ou seja, quando abrir a página não aparecerá nada, após cinco segundos, a função será chamada e o texto “Canal Fessor Bruno” será mostrado.

Como foi dito, setTimeout só chama a função callback uma vez, então, a função “cfb” só será chamada uma única vez.

Podemos configurar a função setTimeout de modo recursivo, desta maneira, a função será chamada várias vezes, mas este é outro assunto.

Caso seja necessário chamar a função callback mais de uma vez, usamos outra função que veremos a seguir.

## setInterval

A função setInterval tem uma pequena diferença de setTimeout, que é chamar a função callback várias vezes, desta maneira podemos criar um “timer” que se repetirá sempre e não somente uma única vez.

Vamos usar um script semelhante ao anterior, simplesmente trocando para “setInterval”.

```
<!doctype html>
<html lang="pt-br">
```

```
<head>
  <title>Curso de Javascript</title>
  <meta charset="UTF-8">
  <script>

    var timer=setInterval(cfb,5000);

    function cfb(){
      document.write("Canal Fessor Bruno");
    }

  </script>
</head>
<body>

</body>
</html>
```

A diferença deste script usando “setInterval” para o anterior é que a função “cfb” será chamada sempre, de cinco em cinco segundos.

Salve e abra a página, note que o texto “Canal Fessor Bruno” será impresso de cinco em cinco segundos.

### *clearInterval*

E se for necessário parar o setInterval? É simples, basta usar a função clearInterval.

A sintaxe é bem simples, veja.

clearInterval(intervalo);

Vamos alterar o script anterior, para que o intervalo seja parado quando completar 10 iterações e vamos diminuir o tempo para 1 segundo.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

      var cont=0;

      var timer=setInterval(cfb,1000);

      function cfb(){
        document.write("Canal Fessor Bruno<br>");
        cont++;
        if(cont == 10){
          clearInterval(timer);
        }
      }

    </script>
  </head>
  <body>

  </body>
</html>
```

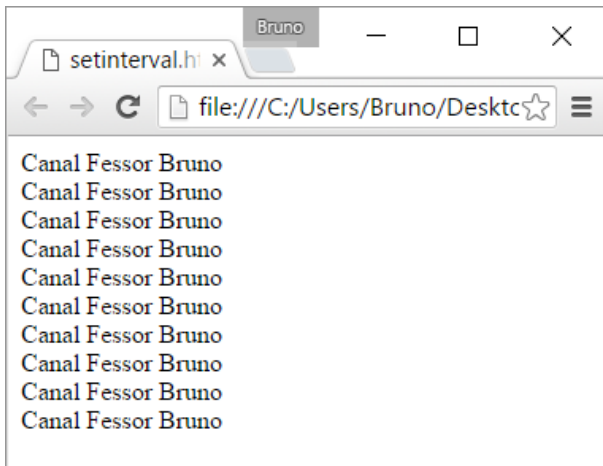
Note que agora temos uma variável “cont” para controlar as iterações que é iniciada com o valor zero.

Dentro da função temos um incremento de um na variável “cont”, ou seja, a cada chamada da função “cfb” a variável “cont” será incrementada em um.

Após o incremento verificamos o valor de “cont” se for igual a 10, chamamos a função clearInterval, com o “variável” que está associada ao interval.

```
function cfb(){
    document.write("Canal Fessor Bruno<br>");
    cont++;
    if(cont == 10){
        clearInterval(timer);
    }
}
```

Veja o resultado final do script.



Vamos criar um script que muda a cor de fundo da página de um em um segundo, com cores definidas em um array.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

        var cores=["#F00","#0F0","#00F","#FF0","#0FF","#F0F","#000","#FFF"];
        var i=0;

        var timer=setInterval(mudaCorFundo,1000);

        function mudaCorFundo(){
            document.body.style.backgroundColor=cores[i];
            i++;
            if(i >= cores.length){
                i=0;
            }
        }

    </script>
  </head>
  <body>

  </body>
</html>
```

Podemos ainda controlar o intervalo por botões, vamos alterar nosso código adicionando dois botões para que inicie e pare o intervalo e de acordo com estes botões.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
```



```

var cores=["#F00","#0F0","#00F","#FF0","#0FF","#F0F","#000","#FFF"];
var i=0;
var timer;

function mudaCorFundo(){
    document.body.style.backgroundColor=cores[i];
    i++;
    if(i >= cores.length){
        i=0;
    }
}

function iniciar(){
    clearInterval(timer);
    timer=setInterval(mudaCorFundo,1000);
}

function parar(){
    clearInterval(timer);
    document.getElementById("bt_ini").value="Continuar troca";
}

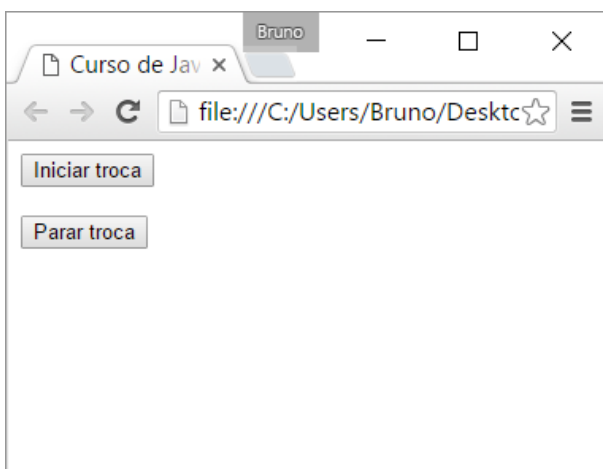
</script>
</head>
<body>

    <input type="button" value="Iniciar troca" id="bt_ini" onclick="iniciar()"><br><br>
    <input type="button" value="Parar troca" onclick="parar()">

</body>
</html>

```

Agora temos dois botões com o evento “onclick” programados para chamar as funções “iniciar()” e “parar()”. Note que quando pressionamos o botão “Parar troca” o texto do outro botão muda de “Iniciar troca” para “Continuar troca”.



## *setInterval como enterFrame*

Lembra do script que fizemos para movimentar o quadrado preto? Podemos melhorar muito o funcionamento deste script usando setInterval, podemos conseguir um movimento mais fluido e aumentar muito nossas possibilidades.

Vamos realizar esta alteração.

```

<!doctype html>
<html lang="pt-br">
    <head>
        <title>Curso de Javascript</title>
        <meta charset="UTF-8">
        <style>
            #jogador{
                position:absolute;

```

```

        left:0px;
        top:0px;
        width:50px;
        height:50px;
        background-color:#000;
    }
</style>
<script>
    var dx=0;
    var dy=0;
    var px=0;
    var py=0;
    var jg=document.getElementById("jogador");

    var intervalo = setInterval(enterFrame, 20);

    function enterFrame(){
        px+=dx*10;
        py+=dy*10;
        jg=document.getElementById("jogador");
        jg.style.left=px+"px";
        jg.style.top=py+"px";
    }

    function load(){
        document.addEventListener("keydown", teclaDw);
        document.addEventListener("keyup", teclaUp);
    }

    function teclaDw(){
        var tecla = event.keyCode;
        if(tecla==37){
            //Esquerda
            dx=-1;
        }else if (tecla==39){
            //Direita
            dx=1;
        }
        if(tecla==38){
            //Cima
            dy=-1;
        }else if (tecla==40){
            //baixo
            dy=+1;
        }
    }

    function teclaUp(){
        var tecla = event.keyCode;
        if(tecla==37){
            //Esquerda
            dx=0;
        }else if (tecla==39){
            //Direita
            dx=0;
        }
        if(tecla==38){
            //Cima
            dy=0;
        }else if (tecla==40){
            //baixo
            dy=0;
        }
    }

    window.addEventListener("load", load);

</script>
</head>
<body>

    <div id="jogador"></div>

</body>
</html>

```

Você já deve ser capaz de entender todo o script acima, tende entender antes de continuar.

Os primeiros comandos inseridos foram as variáveis para controlar a posição X e Y e as variáveis para controlar a direção X e Y.

```
var dx=0;
```

```
var dy=0;
```

```
var px=0;
var py=0;
```

Em sequência as variáveis associadas ao “quadrado preto” <div id= “jogador”> e para o intervalo “setInterval”.

```
var jg=document.getElementById("jogador");
var intervalo = setInterval(enterFrame, 20);
```

Note que nosso setInterval chama a função com nome “enterFrame” de 20 em 20 milésimos de segundo, 0,02 segundos, muito rápido.

A função “enterFrame” simplesmente calcula as novas posições X e Y e passa para as propriedades left e top do quadrado “jogador”.

```
function enterFrame(){
    px+=dx*10;
    py+=dy*10;
    jg=document.getElementById("jogador");
    jg.style.left=px+"px";
    jg.style.top=py+"px";
}
```

A seguir adicionamos os eventos “keydown” para interceptar as teclas pressionadas e “keyup” para interceptar quais as tecla que já foram liberadas.

```
function load(){
    document.addEventListener("keydown",teclaDw);
    document.addEventListener("keyup",teclaUp);
}
```

As funções teclaDw() e teclaUp(), simplesmente definem a direção para 1, -1 ou 0 das variáveis dx e dy de acordo com a tecla pressionada ou liberada.

Por último adicionamos o evento “load” para adicionar os eventos “keydown” e “keyup”.

```
window.addEventListener("load", load);
```

Salve seu código e faça o teste, veja que o quadrado se move de uma maneira muito mais fluida inclusive na diagonal.

## *Criando um relógio digital*

Vamos juntar alguns recursos e criar um relógio digital interessante para ser mostrado em nossas páginas, vamos obter a hora do seu computador e mostrar em uma caixa de texto, depois de tudo pronto você ainda pode formatar tudo com CSS para que fique com uma aparência mais interessante, próxima a um mostrador de relógio digital.

Nosso relógio é bem simples, embora pareça ser um trabalho complicado.

Vamos usar os seguintes recursos, setInterval, function, métodos da classe Date, getElementById, if.

Veja a seguir o código completo para o funcionamento do relógio e logo depois as explicações.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
```

```

<script>

    function relógio(){
        var data=new Date();
        var hor=data.getHours();
        var min=data.getMinutes();
        var seg=data.getSeconds();

        if(hor < 10){
            hor="0"+hor;
        }
        if(min < 10){
            min="0"+min;
        }
        if(seg < 10){
            seg="0"+seg;
        }

        var horas=hor + ":" + min + ":" + seg;

        document.getElementById("rel").value=horas;
    }

    var timer=setInterval(relógio,1000);

</script>
</head>
<body>
    <input type="text" id="rel">
</body>
</html>

```

Primeiro vamos falar sobre o `<input>`, veja que no `<body>` nós temos inserido um elemento `<input type="text" id="rel">`, esta caixa de texto receberá as informações referente às horas obtidas do seu computador e devidamente configuradas para o formato "hh:mm:ss".

No script o primeiro passo é criar a função que irá "preparar" o relógio, demos o nome para esta função de "relógio()", vamos entender como ele funciona.

```
function relógio(){
```

Primeiramente na função "relógio()" criamos o objeto data do tipo Date e obtivemos as horas, minutos e segundos devidamente armazenados nas variáveis hor, min, seg.

```

var data=new Date();
var hor=data.getHours();
var min=data.getMinutes();
var seg=data.getSeconds();

```

Estas funções retornam os valores menores que 10 com um dígito só, no caso de relógios, estes valores são mostrados com dois dígitos, com o zero na frente, 01, 02, ..., 09, então, precisamos "ajustar" estes valores, vamos usar IF para testar se o valor é menor que dez, se for, vamos adicionar o "0" na frente do valor.

```

if(hor < 10){
    hor="0"+hor;
}
if(min < 10){
    min="0"+min;
}
if(seg < 10){
    seg="0"+seg;
}

```

Em seguida vamos juntar todas as informações de horas, minutos e segundos em uma só variável, no caso, a variável “horas”.

```
var horas=hor + ":" + min + ":" + seg;
```

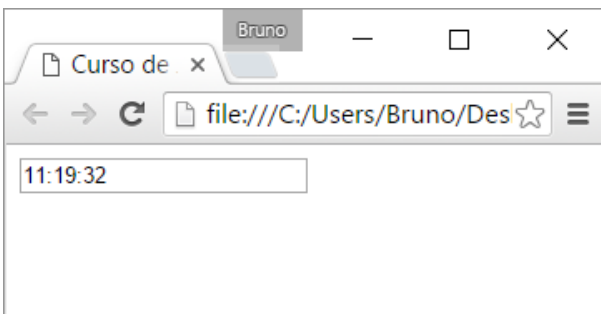
O último passo da função é adicionar a hora devidamente preparada que está na variável “horas” no <input>.

```
document.getElementById("rel").value=horas;
```

Com a função pronta, basta criar o timer “setInterval” que irá chamar a função “relogio()” de um em um segundo.

```
var timer=setInterval(relogio,1000);
```

Pronto, agora já temos um relógio analógico funcionando, basta formatar para ter a aparência que desejar.



## Recursividade

Recursividade é um assunto que espanta muitas pessoas, mas é simples, basicamente você precisa entender o que é recursividade? A definição mais simples de recursividade é quando temos uma função que chama a si mesma, entrando em uma forma de loop, como se fosse um FOR, precisamos ter muito cuidado ao programar funções recursivas, pela condição de parada que não é tão óbvia quanto o FOR.

Vamos criar uma função simples que faz a uma contagem e impressão de valores, vamos passar para esta função o valor inicial e final da contagem.

Vamos ao código e depois a explicação.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

      function contador(cont,num){
        document.write(cont + "<br>");
        if(num > cont){
          contador(++cont,num);
        }
      }

      contador(0,20);
    </script>
  </head>
  <body>

  </body>
</html>
```

Primeiramente criamos a função com dois parâmetros de entrada, cont que receberá o valor inicial da contagem e num que receberá o valor final da contagem.

```
function contador(cont,num){
```

Dentro da função, a primeira coisa a ser feita é imprimir o valor de “cont”.

```
document.write(cont + "<br>");
```

Em seguida vamos programar o IF que irá verificar a condição para a nova chamada da função, se a condição do IF for satisfeita (true) a função é chamada novamente, caso contrário a função não é chamada e tudo é encerrado.

```
if(num > cont){  
    contador(++cont,num);  
}
```

Note que o conte é incrementado como pré-incremento e não pós-incremento, pois, aqui precisamos entrar com a variável cont já incrementada.

Para ativar a recursividade chamamos a função passando os valores inicial e final da contagem, no nosso exemplo, iniciamos a contagem em zero e terminamos em 20.

```
contador(0,20);
```

Vou detalhar o funcionamento desta recursividade.

Iniciamos a chamada da função com os valores cont=0 e num=20;

A função então verifica, Se num é maior que cont, como é maior e este teste retorna true, simplesmente chama a função contador novamente, porém, com o valor de cont incrementado em um.

Então chama a função contador com os valores cont=1 e num=20, assim o processo é repetido, até que conta tenha o valor 20, neste momento o teste irá retornar false e não chamará a função novamente, saindo da recursividade.

Vamos ver outro exemplo de recursividade usando setTimeout.

O próximo exemplo de recursividade imprime o texto “Canal Fessor Bruno” a cada meio segundo.

```
<!doctype html>  
<html lang="pt-br">  
  <head>  
    <title>Curso de Javascript</title>  
    <meta charset="UTF-8">  
    <script>  
  
        var timer;  
  
        function cfb(){  
            document.write("Canal Fessor Bruno<br>");  
            timer=setTimeout(cfb,500);  
        }  
  
        cfb();  
  
    </script>  
  </head>  
  <body>  
  
  </body>  
</html>
```

Lembra-se de setTimeout? Lembra-se que faz a chamada da função callback somente uma vez? Usando recursividade conseguimos configurar para que funcione de forma semelhante à setInterval.

Podemos limpar este timer setTimeout usando clearTimeout, vamos alterar nosso código de forma que sejam impressos somente 10 textos e limpe o intervalo.

```
<script>
```

```

var timer;
var cont=0;

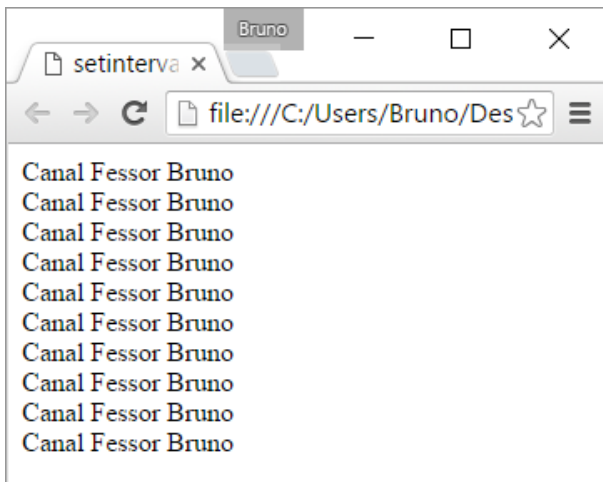
function cfb(){
    if(cont <= 10){
        cont++;
        document.write("Canal Fessor Bruno<br>");
        timer=setTimeout(cfb, 500);
    }
}

cfb();

</script>

```

Após a impressão de 10 linhas de texto o intervalo “timer” será cancelado.



Vamos praticar um pouco mais sobre recursividade.

Criaremos a seguir uma função recursiva que irá calcular o fatorial de um número, lembra-se o que é fatorial?

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

$$8! = 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 40320$$

Vamos ao passo a passo do processo para cálculo do fatorial de cinco para podermos criar a função recursiva.

$$5 \times 4 = 20$$

$$20 \times 3 = 60$$

$$60 \times 2 = 120$$

$$120 \times 1 = 120$$

De outra ponto de vista

$$5 \times 4 = 20 \times 3 = 60 \times 2 = 120 \times 1 = 120$$

Então como podemos transformar este processo em uma função recursiva?

Precisamos de passar para a função o valor inicial, o cinco no nosso exemplo.

Dentro da função precisamos multiplicar o resultado pelo valor anterior, o quatro. No próximo passo a mesma coisa, multiplicar este resultado pelo valor anterior, o três e assim por diante até que o valor seja um para indicar o fim do processo.

Veja o código.

```

<!doctype html>
<html lang="pt-br">

```

```

<head>
  <title>Curso de Javascript</title>
  <meta charset="UTF-8">
  <script>
    var num=8;

    function fatorial(n){
      if(n == 1){
        return 1;
      }
      return n*fatorial(n-1);
    }

    document.write("Fatorial de " + num + ": " + fatorial(num));
  </script>
</head>
<body>

</body>
</html>

```

Criamos a função “fatorial” recebendo um valor como parâmetro “n”.

```
function fatorial(n){
```

Em seguida, verificamos se o valor de “n” já é igual a 1 para que simplesmente retorne 1 e não chame a função fatorial.

```

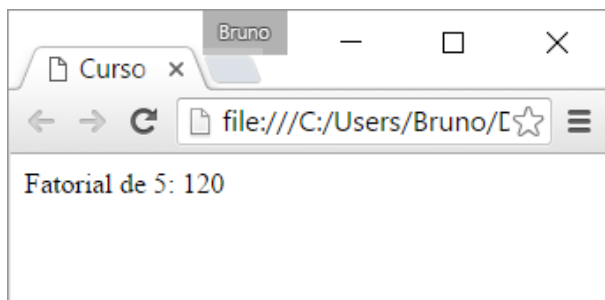
if(n == 1){
  return 1;
}

```

Aqui está o processo mais importante, a recursividade funcionando. O retorno da função fatorial é a multiplicação do valor passado como parâmetro “n” pelo retorno da função fatorial com o valor de “n-1”, ou seja, na primeira iteração, é 5 x 4.

```
return n*fatorial(n-1);
```

Como o valor de retorno da primeira iteração é 20 então é multiplicado este retorno por n-1 e assim por diante.



Pronto criamos uma função recursiva para calcular o fatorial de um número.

Podemos modificar esta função de forma simples para calcular o somatório de um determinado número, vejamos.

$$S = \sum_{i=0}^{10} i \rightarrow 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 55$$

Vamos ao código.

```

<script>
  var num=10

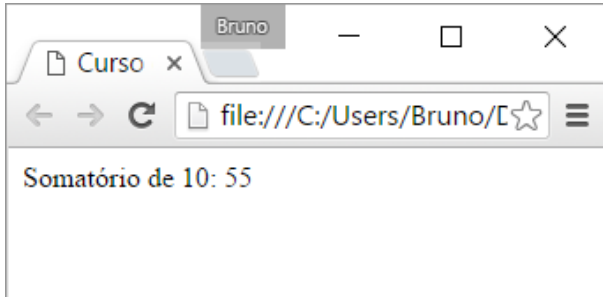
  function soma(n){

```



```
    if(n == 1){
        return 1;
    }
    return n+soma(n-1);
}

document.write("Somatório de " + num + ": " + soma(num));
</script>
```



## Validação de formulários

Neste capítulo vamos aprender um pouco sobre como realizar a checagem dos campos de um formulário para fazer a validação dos dados digitados, não iremos criar uma rotina avançada para validação, vamos construir um código de validação simples, pois, o objetivo deste capítulo é mostrar como podemos verificar os campos dos formulários, você pode implementar suas próprias regras de validação como desejar.

O primeiro passo é criar o formulário, criaremos um básico com quatro campos de texto e um botão submit, veja o código a seguir.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

    </script>
  </head>
  <body>

    <form name="f_aula" method="#" action="#">

      <label>Nome:<label><br>
      <input type="text" name="f_nome" id="f_inome" maxlength="50"><br><br>

      <label>Senha:<label><br>
      <input type="password" name="f_senha" id="f_isenha" maxlength="20"><br><br>

      <label>Data Nascimento:<label><br>
      <input type="text" name="f_data" id="f_idata" maxlength="10"><br><br>

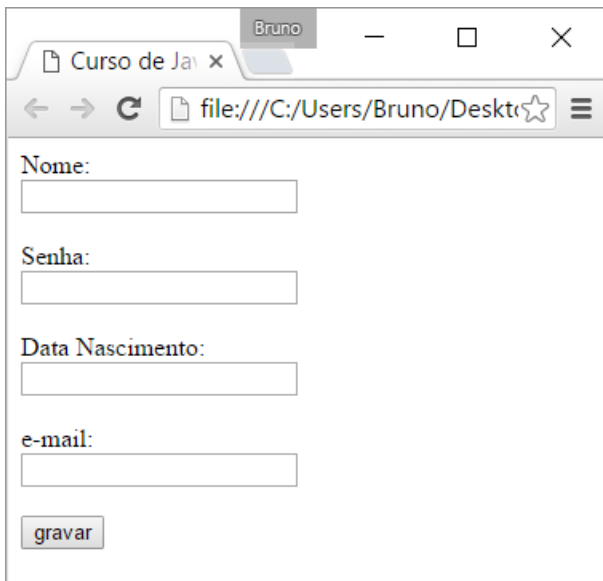
      <label>e-mail:<label><br>
      <input type="text" name="f_email" id="f_iemail" maxlength="50"><br><br>

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

    </form>

  </body>
</html>
```

O código acima resulta no formulário a seguir.



Com o formulário pronto podemos criar nosso script de validação. A validação será feita quando o formulário for submetido, ou seja, no evento “onsubmit”, quando clicar no botão “gravar”.

Vamos criar a função e associa-la ao evento “onsubmit”.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

      function valida() {

      }

    </script>
  </head>
  <body>

    <form name="f_aula" method="#" action="#" onsubmit="return valida()">

      <label>Nome:</label><br>
      <input type="text" name="f_nome" id="f_inome" maxlength="50"><br><br>

      <label>Senha:</label><br>
      <input type="password" name="f_senha" id="f_isenha" maxlength="20"><br><br>

      <label>Data Nascimento:</label><br>
      <input type="text" name="f_data" id="f_idata" maxlength="10"><br><br>

      <label>e-mail:</label><br>
      <input type="text" name="f_email" id="f_iemail" maxlength="50"><br><br>

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

    </form>

  </body>
</html>
```

Note que no evento “onsubmit” usamos o retorno da função valida(). Então, se a função retornar “false” o evento é interrompido e o formulário não é enviado.

Na função “valida()” vamos criar a primeira rotina de validação para o campo nome, iremos simplesmente verificar se o campos está preenchido ou não.

```
<script>

function valida(){

    if(document.f_aula.f_nome.value == ""){
        alert("Digite o nome");
        document.getElementById("f_inome").focus();
        return false;
    }

}

</script>
```

O IF verifica se o campo não possui nenhum valor, ou seja, se seu conteúdo é uma string vazia, se for, é mostrada uma mensagem alert, o campo “f\_inome” recebe o cursor “focus()”, e o retorno da função é falso.

```
if(document.f_aula.f_nome.value == ""){
    alert("Digite o nome");
    document.getElementById("f_inome").focus();
    return false;
}
```

O próximo campo a ser validado será o campo de senha, da mesma forma que o campo nome.

```
<script>

function valida(){

    if(document.f_aula.f_nome.value == ""){
        alert("Digite o nome");
        document.getElementById("f_inome").focus();
        return false;
    }

    if(document.f_aula.f_senha.value == ""){
        alert("Digite a senha");
        document.getElementById("f_isenha").focus();
        return false;
    }

}

</script>
```

Como não tem novidades nesta rotina em relação à anterior vamos para validação da data.

```
<script>

function valida(){

    if(document.f_aula.f_nome.value == ""){
        alert("Digite o nome");
        document.getElementById("f_inome").focus();
        return false;
    }

    if(document.f_aula.f_senha.value == ""){
        alert("Digite a senha");
        document.getElementById("f_isenha").focus();
        return false;
    }

    var data=new Date();
    var dt=new Array;
    var aux=document.f_aula.f_data.value;
    dt=aux.split("/");

    if( ((dt[0] < 1)|| (dt[0] > 31)) || ((dt[1] < 1)|| (dt[1] > 12)) || ((dt[2] < dt.getFullYear()-150)|| (dt[2] > dt.getFullYear())) ){
        alert("Digite uma data válida");
        document.getElementById("f_idata").focus();
        return false;
    }

}

</script>
```

Na validação da data temos uma verificação um pouco mais complexa, vamos verificar o valor do dia, do mês e do ano, se estes estiverem em conformidade passamos para frente, se não, emitimos a mensagem, posicionamos o cursor no campo e interrompemos o envio do formulário.

Para validar vamos criar um objeto do tipo "Date()" que será usado na verificação do ano.

```
var data=new Date();
```

Em seguida criamos uma array para que possamos guardar o dia, mês e ano, em cada posição do array, dia na posição [0], mês na posição [1] e ano na posição [2].

```
var dt=new Array;
```

Agora precisamos criar uma variável para receber a data digitada no campo do formulário.

```
var aux=document.f_aula.f_data.value;
```

Com o array e a variável com a data digitada no formulário, podemos usar o método "split" para obter o dia, mês e ano, informamos ao método "split" que o caractere divisor será a barra "/", desta forma, conseguimos separar os valores da data.

Como estamos associando o resultado do "split" ao array, será armazenado na posição [0] os valores da primeira divisão relacionados ao dia, na posição [1] os valores relacionados ao mês e na posição [2] os valores relacionados ao ano.

```
dt=aux.split("/");
```

Em seguida usamos um IF para verificar os valores do array, usamos regras simples:

Dia → O valor deve ser maior ou igual a 1 e menor ou igual a 31.

Mês → O valor deve estar entre 1 e 12.

Ano → O valor deve ser maior o igual ao ano atual menos 150 e menor ou igual ao ano atual.

```
if( ((dt[0] < 1) || (dt[0] > 31)) || ((dt[1] < 1) || (dt[1] > 12)) || ((dt[2] < dt.getFullYear()-150) || (dt[2] > dt.getFullYear())) ){
```

Em seguida vêm os comandos caso a data não seja validada.

```
alert("Digite uma data válida");
document.getElementById("f_idata").focus();
return false;
```

A última validação será do e-mail.

```
<script>

function valida(){

    if(document.f_aula.f_nome.value == ""){
        alert("Digite o nome");
        document.getElementById("f_inome").focus();
        return false;
    }

    if(document.f_aula.f_senha.value == ""){
        alert("Digite a senha");
        document.getElementById("f_isenha").focus();
        return false;
    }

    var data=new Date();
    var dt=new Array;
    var aux=document.f_aula.f_data.value;
    dt=aux.split("/");
```

```

    if( ((dt[0] < 1)|| (dt[0] > 31)) || ((dt[1] < 1)|| (dt[1] > 12)) || ((dt[2] < dt.getFullYear()-150)|| (dt[2] > dt.getFullYear())) ){
        alert("Digite uma data válida");
        document.getElementById("f_idata").focus();
        return false;
    }

    var vm=document.f_aula.f_email.value;
    if (vm.search("@")==-1){
        alert("Digite um e-mail válido");
        document.getElementById("f_iemail").focus();
        return false;
    }
}
</script>

```

Neste procedimento vamos armazenar o e-mail digitado na variável “vm”.

```
var vm=document.f_aula.f_email.value;
```

Em seguida vamos simplesmente verificar se o e-mail digitado possui o caractere “@”, vamos usar o método “search” para isto, caso o método não encontre o caractere especificado é retornado o valor “-1”, então, basta verificar o retorno do método, se for igual a -1 significa que o caractere não foi encontrado.

```

if(vm.search("@")==-1){
    alert("Digite um e-mail válido");
    document.getElementById("f_iemail").focus();
    return false;
}

```

Salve as alterações e faça alguns testes com o formulário tentando gravar sem preencher alguns campos, teste o campo de data com uma data inválida segundo as nossas especificações e o campo e-mail sem o @.

## JavaScript x CSS

Podemos manipular as propriedades de formatação CSS pelo javascript de forma muito simples, esta é uma possibilidade bastante interessante que aumenta muito nosso leque de possibilidades com javascript, neste capítulo iremos aprender como usar CSS pelo javascript e ao final terá uma tabela com uma grande variedade de propriedades para consulta.

Observe o código a seguir.

```

<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

        document.write("<p id=curso>Curso de Javascript</p>");

        document.getElementById("curso").style.color="#F00";

    </script>
  </head>
  <body>

    <p id="cfb">Canal Fessor Bruno</p>

  </body>
</html>

```

O resultado deste código é mostrado na ilustração a seguir.



Note que inserimos um primeiro parágrafo via javascript com o método “write” e no corpo do <body> inserimos um segundo parágrafo.

Ainda no script, logo após o “write”, formatamos a cor do texto referente ao id “curso” para vermelho.

Veja como é simples entender.

```
document.getElementById("curso").style.color="#F00";
```

Primeiro nos referimos ao elemento que desejamos trabalhar, indicamos este elemento pelo ID.

Depois indicamos que desejamos trabalhar com a propriedade “style” e em seguida o nome do estilo que desejamos alterar. Por último atribuímos um valor ao estilo indicado.

E se precisarmos alterar a cor do segundo parágrafo para azul?

Neste caso, não basta adicionar o código de formatação de forma tão simples quando ao anterior, veja.

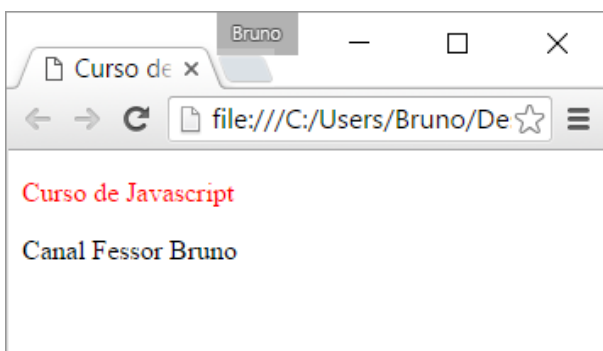
```
<script>

    document.write("<p id=curso>Curso de Javascript</p>");

    document.getElementById("curso").style.color="#F00";
    document.getElementById("cfb").style.color="#00F";

</script>
```

Note que mesmo adicionando o comando de formatação o texto não será formatado.



Isso acontece devido ao carregamento do <p> ser posterior ao carregamento do código de formatação, então como resolver isto?

Na verdade já sabemos esta resposta.

Devemos chamar o comando de formatação no “load” da página, veja as alterações.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
```

```

<meta charset="UTF-8">
<script>

    document.write("<p id=curso>Curso de Javascript</p>");

    document.getElementById("curso").style.color="#F00";

    function formata(){
        document.getElementById("cfb").style.color="#00F";
    }

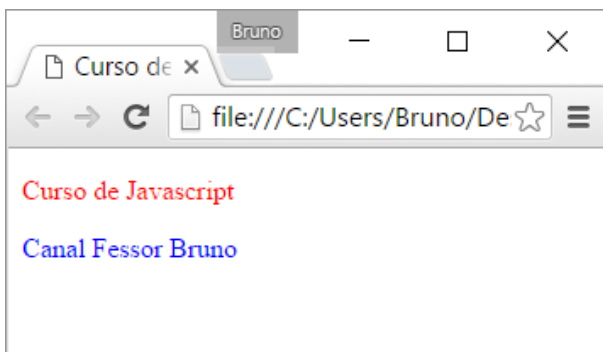
</script>
</head>
<body onload="formata()" ">

    <p id="cfb">Canal Fessor Bruno</p>

</body>
</html>

```

Assim conseguimos formatar o segundo parágrafo, veja a imagem.



Viu como é simples!

A seguir disponho uma tabela com uma série de propriedades CSS que podem ser manipuladas por javascript, na esquerda temos a propriedade original CSS e no centro como deve ser usada no javascript, na coluna da direita tem uma breve descrição da propriedade.

CSS	JavaScript	Descrição
		*Os valores sublinhados são os padrões. *& indica que as propriedades anteriores se repetem após o valor indicado.
align-content	alignContent	Alinhamento entre as linhas dentro de um container flexível quando os itens não usam todo o espaço disponível. Valores: <u>stretch</u> , center, flex-start, flex-end, space-between, space-around, initial, inherit.
align-items	alignItems	Alinhamento para itens dentro de um container flexível. Valores: <u>stretch</u> , center, flex-start, flex-end, space-between, space-around, baseline, initial, inherit.
align-self	alignSelf	Definir o alinhamento de um dos itens no interior de um elemento flexível para se ajustar ao container. Valores: <u>auto</u> , stretch, center, flex-start, flex-end, space-between, space-around, baseline, initial, inherit.
animation	animation	Metapropriedade. Define todas as propriedades de animação a seguir, menos animationPlayState.
animation-delay	animationDelay	Define quando a animação vai começar. Valores: time, initial, inherit. (padrão time com valor 0)
animation-direction	animationDirection	Direção da animação. Valores: <u>normal</u> , reverse, alternate, alternate-reverse, initial, inherit
animation-duration	animationDuration	Tempo de duração da animação em milissegundos. Valores: time, initial, inherit. (padrão time com valor 0)
animation-fill-mode	animationFillMode	Quais valores serão aplicadas pela animação fora do período que ele está executando. Valores: <u>none</u> , forwards, backwards, both, initial, inherit

animation-iteration-count	animationIterationCount	Número de vezes que uma animação deve ser executada. Valores: number, infinite, initial, inherit (padrão number com valor 1)
animation-name	animationName	Nome da animação ou para os @keyframes da animação. Valores: <u>none</u> , keyframename, initial, inherit
animation-timing-function	animationTimingFunction	Curva de velocidade da animação. Valores: linear, <u>ease</u> , ease-in, ease-out, cubic-bezier(n,n,n,n), initial, inherit
animation-play-state	animationPlayState	Define se a animação está em execução ou em pausa. Valores: <u>running</u> , paused, initial, inherit
azimuth	azimuth	Posicionamento espacial do áudio gerado pelo objeto. Valores: ângulo, left-side, far-left, left, center-left, <u>center</u> , center-right, right, far-right, right-side, [& behind], leftwards, rightwards, inherit
background	background	Metapropriedade. Define backgroundColor, backgroundImage, backgroundRepeat, backgroundAttachment e backgroundPosition.
background-attachment	backgroundAttachment	Indica se a imagem de fundo vai ser fixa ou vai rolar. Valores: <u>scroll</u> , fixed, inherit
background-color	backgroundColor	Cor de fundo. Valores: cor, <u>transparente</u> , inherit
background-image	backgroundImage	Imagem de fundo. Valores: URL, <u>none</u> , inherit
background-position	backgroundPosition	Cordenadas X e Y da imagem de fundo. Valores: porcentagem, tamanho (exemplo: x,y), top, center, bottom, left, center, right, inherit
background-repeat	backgroundRepeat	Configura a repetição da imagem de fundo. Valores: <u>repeat</u> , repeat-x, repeat-y, no-repeat, inherit
background-clip	backgroundClip	Especifica a área de exibição da imagem de fundo. Valores: <u>border-box</u> , padding-box, content-box, initial, inherit
background-origin	backgroundOrigin	Posicionamento de origem da imagem. Valores: <u>padding-box</u> , border-box, content-box, initial, inherit
background-size	backgroundSize	Tamanho da imagem de fundo. Valores: <u>auto</u> , length, cover, contain, initial, inherit
backface-visibility	backfaceVisibility	A propriedade backfaceVisibility define se ou não um elemento deve ser visível quando não está rotacionada de frente para a tela. Esta propriedade é útil quando um elemento é rodado, e você não quer ver a parte de trás. Valores: <u>visible</u> , hidden, initial, inherit
border	border	Metapropriedade. Define todas as bordas ao mesmo tempo.
border-bottom	borderBottom	Metapropriedade. Define as propriedades borderBottomWidth, borderBottomStyle e borderBottomColor.
border-bottom-color	borderBottomColor	Cor da borda inferior. Valores: cor, inherit
border-radius	borderRadius	Define o arredondamento de todas as bordas igualmente Valores: length %, initial, inherit (valor padrão 0)
border-bottom-left-radius	borderBottomLeftRadius	Arredonda a borda inferior esquerda Valores: 0 (% porcentagem), initial, inherit (padrão valor = 0)
border-bottom-right-radius	borderBottomRightRadius	Arredonda a borda inferior direita Valores: 0 (% porcentagem), initial, inherit (padrão valor = 0)
border-top-left-radius	borderTopLeftRadius	Arredonda a borda superior esquerda Valores: 0 (% porcentagem), initial, inherit (padrão valor = 0)
border-top-right-radius	borderTopRightRadius	Arredonda a borda superior direita Valores: 0 (% porcentagem), initial, inherit (padrão valor = 0)
border-bottom-style	borderBottomStyle	Estilo da borda inferior. Valores: <u>none</u> , hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, inherit
border-bottom-width	borderBottomWidth	Largura da borda inferior. Valores: thin, <u>medium</u> , thick, tamanho, inherit
Border-collapse	borderCollapse	Reduzir a borda da tabela Valores: <u>separate</u> , collapse, initial, inherit



border-color	borderColor	Cores de todas as bordas.
border-image	borderImage	Metapropriedade, especifica uma imagem como borda em torno de um elemento <div> Valores: source slice width outset repeat, initial, inherit
border-image-outset	borderImageOutset	Configura a imagem de borda para fora ou para dentro das bordas originais da <div> Valores: length, number, initial, inherit (valor padrão 0), para ajustar internamente use valores menores que 1, por exemplo 0.1
border-image-repeat	borderImageRepeat	Configura a repetição da imagem usada como borda. Valores: <u>stretch</u> , repeat, round, initial, inherit
border-image-slice	borderImageSlice	Especifica o deslocamento/distância das imagens da borda em caso de repetição Valores: number %, fill, initial, inherit (valor padrão 100%)
border-image-source	borderImageSource	Especifica qual imagem será usada como borda em torno de um elemento div: Valores: <u>none</u> , image, initial, inherit
border-image-width	borderImageWidth	Largura da imagem de borda Valores: number %, auto, initial, inherit (valor padrão 1)
border-left	borderLeft	Semelhante a borderBottom aplicado à borda esquerda.
border-left-color	borderLeftColor	Semelhante a borderBottomColor aplicado à borda esquerda.
border-left-style	borderLeftStyle	Semelhante a borderBottomStyle aplicado à borda esquerda.
border-left-width	borderLeftWidth	Semelhante a borderBottomWidth aplicado à borda esquerda.
border-right	borderRight	Semelhante a borderBottom aplicado à borda direita.
border-right-color	borderRightColor	Semelhante a borderBottomColor aplicado à borda direita.
border-right-style	borderRightStyle	Semelhante a borderBottomStyle aplicado à borda direita.
border-right-width	borderRightWidth	Semelhante a borderBottomWidth aplicado à borda direita.
border-spacing	borderSpacing	Espaçamento entre as bordas de duas células adjacentes, somente para tabelas. Valores: tamanho, inherit.
border-style	borderStyle	Estilo de todas as bordas. Valores: <u>none</u> , hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, inherit
border-top	borderTop	Semelhante a borderBottom aplicado à borda esquerda.
border-top-color	borderTopColor	Semelhante a borderBottomColor aplicado à borda esquerda.
border-top-style	borderTopStyle	Semelhante a borderBottomStyle aplicado à borda esquerda.
border-top-width	borderTopWidth	Semelhante a borderBottomWidth aplicado à borda esquerda.
border-width	borderWidth	Largura de todas as bordas. Valores: thin, <u>medium</u> , thick, tamanho
bottom	bottom	Distância entre a base de um objeto e a base do bloco onde está contido. Valores: tamanho, porcentagem, <u>auto</u> , inherit
box-shadow	boxShadow	Adiciona sombra a um elemento div. Valores: <u>none</u> , h-shadow v-shadow blur spread color, inset, initial, inherit
box-sizing	boxSizing	Permite definir certos elementos para caber dentro de em um outro elemento. Valores: <u>content-box</u> , border-box, initial, inherit
caption-side	captionSide	Define ou retorna a posição do table caption. Valores: <u>top</u> , bottom, initial, inherit
clear	clear	Com o valor configurado diferente de none, será posicionado abaixo dos elementos flutuantes que estiverem alinhados da maneira especificada aqui. Valores: <u>none</u> , left, right, both, inherit
clip	clip	Área visível de um objeto. Valores: forma, <u>auto</u> , inherit.
color	color	Cor do texto. Valores: cor, inherit

column-count	columnCount	Especifica o número de colunas que um elemento pode conter Valores: number, auto, initial, inherit
column-gap	columnGap	Especifica a lacuna, separação entre as colunas Valores: length, <u>normal</u> , initial, inherit
column-rule	columnRule	Metapropriedade para definir todas as configurações de columnRule
column-rule-color	columnRuleColor	Especifica a cor da barra divisória entre as colunas Valores: color, initial, inherit
column-rule-style	columnRuleStyle	Define o estilo entre as colunas Valores: <u>none</u> , hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, initial, inherit
column-rule-width	columnRuleWidth	Largura da separação entre as colunas Valores: <u>medium</u> , thin, thick, length, initial, inherit
columns	columns	Número de colunas Valores: <u>auto</u> , column-width column-count, initial, inherit
column-span	columnSpan	Especifica quantas colunas um elemento deve abranger Valores: 1, all, initial, inherit
column-width	columnWidth	Largura da coluna Valores: <u>auto</u> , length, initial, inherit
cursor	cursor	Estilo do ponteiro do Mouse. Valores: url, <u>auto</u> , crosshair, default, pointer, move, e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize, text, wait, help, inherit.
direction	direction	Direção do texto de um objeto e a ordem de construção das colunas de uma tabela. Valores: <u>ltr</u> , rtl, inherit
display	display	Forma de visualização do objeto. Valores: <u>inline</u> , block, list-item, run-in, compact, marker, table, inline-table, table-row-group, table-header-group, table-footer-group, table-row, table-column-group, table-column, table-cell, table-caption, none, inherit.
empty-cells	emptyCells	Em tabelas que usam conteúdos separador por células, configura se as células sem conteúdo terão bordas ou não. Valores: <u>show</u> , hide, inherit
filter	filter	Aplica filtros de sombra, iluminação, etc Valores: <u>none</u>   blur()   brightness()   contrast()   drop-shadow()   grayscale()   hue-rotate()   invert()   opacity()   saturate()   sepia()
flex	flex	Configura todos os itens flexíveis com a mesma largura Valores: flex-grow flex-shrink flex-basis, auto, initial, inherit
flex-basis	flexBasis	Configura a largura inicial de item flexível Valores: number, <u>auto</u> , initial, inherit
flex-direction	flexDirection	Configura a direção de itens flexíveis Valores: row, row-reverse, column, column-reverse, initial, inherit
flex-flow	flexFlow	Fluxo dos elementos flexíveis Valores: flex-direction, flex-wrap, initial, inherit
flex-grow	flexGrow	Especifica quanto o item irá crescer em relação ao resto dos elementos flexíveis no interior do mesmo container Valores: number, initial, inherit
flex-shrink	flexShrink	Especifica quanto o item irá diminuir em relação ao resto dos elementos flexíveis no interior do mesmo container Valores: number, initial, inherit
flex-wrap	flexWrap	Especifica se os itens flexíveis deve envolver ou não Valores: nowrap, wrap, wrap-reverse, initial, inherit
float	cssFloat	Especifica se o elemento vai flutuar a direita ou a esquerda Valores: left, right, <u>none</u> , initial, inherit
font	font	Metapropriedade. Configura todas as propriedades da fonte. Valores: fontStyle   fontVariant   fontWeight   fontSize   lineHeight   fontFamily, caption, icon, menu, message-box, small-caption, status-bar, inherit
font-family	fontFamily	Define uma ou mais famílias de fonte. Valores: [fonte1, fonte2], inherit
font-size	fontSize	Tamanho da fonte.

		Valores: tamanho, porcentagem, xx-small, x-small, small, <u>medium</u> , large, x-large, xx-large, larger, smaller, inherit.
font-style	fontStyle	Estilo da fonte Valores: <u>normal</u> , italic, oblique, initial, inherit
font-variant	fontVariant	Como as letras da fonte vão ser exibidas Valores: <u>normal</u> , small-caps, initial, inherit
font-stretch	fontStretch	Largura dos caracteres da fonte, wider estica e narrower achata a fonte. Valores: <u>normal</u> , wider, narrower, ultra-condensed, extra-condensed, condensed, semi-condensed, semi-expanded, expanded, extra-expanded, ultra-expands, inherit.
font-weight	fontWeight	Espessura da fonte. Valores: <u>normal</u> , bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900, inherit
font-stretch	fontStretch	Especifica como será o ajuste da fonte, se irá esticar ou encolher por exemplo Valores: <u>normal</u> , condensed, expanded
height	height	Altura do objeto. Valores: tamanho, porcentagem, <u>auto</u> , inherit.
image-orientation	imageOrientation	Especifica a orientação/direção da imagem
left	left	Semelhante à propriedade bottom, porém, aplicado à esquerda. Valores: tamanho, porcentagem, <u>auto</u> , inherit
letter-spacing	letterSpacing	Distância entre os caracteres do texto. Valores: <u>normal</u> , tamanho, inherit
line-height	lineHeight	Altura da linha. Valores: <u>normal</u> , fator, tamanho, porcentagem, inherit.
list-style	listStyle	Metapropriedade. Define as propriedades listStyleType, listStylePosition e listStyleImage.
list-style-image	listStyleImage	Imagem que será usada como marcador. Valores: url, <u>none</u> , inherit
list-style-position	listStylePosition	Posição do marcador. Valores: inside, <u>outside</u> , inherit
list-style-type	listStyleType	Tipo do marcador. Valores: <u>disc</u> , circle, square, decimal, decimal-leading-zero, lower-roman, upper-roman, lower-greek, lower-alpha, lower-latin, upper-alpha, upper-latin, hebrew, armenian, georgian, cjk-ideographic, hiragana, katakana, hiragana-iroha, katakana-iroha, none, inherit
margin	margin	Largura das quatro margens. Valores: tamanho (exemplo: 1,2,3,4), inherit
margin-bottom	marginBottom	Largura da margem inferior. Valores: tamanho, inherit
margin-left	marginLeft	Largura da margem esquerda. Valores: tamanho, inherit
margin-right	marginRight	Largura da margem direita. Valores: tamanho, inherit
margin-top	marginTop	Largura da margem superior. Valores: tamanho, inherit
max-height	maxHeight	Altura máxima do elemento Valores: none, px, %, initial, inherit
max-width	maxWidth	Largura máxima do elemento Valores: none, px, %, initial, inherit
min-height	minHeight	Altura mínima do elemento Valores: none, px, %, initial, inherit
min-width	minWidth	Largura mínima do elemento Valores: none, px, %, initial, inherit
nav-down	navDown	Define ou retorna para onde navegar ao usar a tecla de navegação seta-para baixo
nav-index	navIndex	Ordem de tabulação de um elemento
nav-left	navLeft	Comportamento ao usar a tecla de navegação de seta-esquerda

nav-right	navRight	Comportamento ao usar a tecla de navegação de seta-direita
nav-up	navUp	Comportamento ao usar a tecla de navegação de seta-para cima
opacity	opacity	Transparência do elemento <div> Valores: number, initial, inherit
order	order	Ordem de itens flexíveis Valores: number,  initial, inherit
orphans	orphans	Especifica o número mínimo de linhas que serão mostradas no final da página Valores: number, initial, inherit
outline	outline	Metapropriedade. Define as propriedades de contorno, outlineColor, outlineStyle, outlineWidth, se difere da borda por não ocupar espaço e não precisa ser retangular.
outline-color	outlineColor	Cor do contorno. Valores: cor, <u>invert</u> , inherit
outline-style	outlineStyle	Estilo do contorno. Valores: <u>none</u> , hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, inherit
outline-width	outlineWidth	Largura do contorno. Valores: thin, <u>medium</u> , thick, largura, inherit.
overflow	overflow	Define o comportamento visual do objeto caso ele ultrapasse a área definida pelo bloco pai. Valores: <u>visible</u> , hidden, scroll, auto, inherit
overflow-x	overflowX	Define o comportamento visual do objeto caso ele ultrapasse a área definida pelo bloco pai pela horizontal Valores: <u>visible</u> , hidden, scroll, auto, initial, inherit
overflow-y	overflowY	Define o comportamento visual do objeto caso ele ultrapasse a área definida pelo bloco pai pela vertical Valores: <u>visible</u> , hidden, scroll, auto, initial, inherit
padding	padding	Todos os lados do espaçamento interno. Valores: tamanho (ex: 1,2,3,4), porcentagem (ex: 1,2,3,4), inherit
padding-bottom	paddingBottom	Espaçamento interno inferior. Valores: tamanho, porcentagem, inherit.
padding-left	paddingLeft	Espaçamento interno esquerdo. Valores: tamanho, porcentagem, inherit.
padding-right	paddingRight	Espaçamento interno direito. Valores: tamanho, porcentagem, inherit.
padding-top	paddingTop	Espaçamento interno superior. Valores: tamanho, porcentagem, inherit.
page-break-after	pageBreakAfter	Define ou retorna o comportamento page-break depois de um elemento (para impressão ou visualização de impressão). Valores: <u>auto</u> , always, avoid, emptystring, left, right, initial, inherit
page-break-before	pageBreakBefore	Define ou retorna o comportamento page-break antes de um elemento (para impressão ou visualização de impressão). Valores: <u>auto</u> , always, avoid, emptystring, left, right, initial, inherit
page-break-inside	pageBreakInside	Define ou retorna o comportamento page-break dentro de um elemento (para impressão ou visualização de impressão). Valores: <u>auto</u> , avoid, initial, inherit
perspective	perspective	Perspectiva de visualização do elemento Valores: length, <u>none</u>
perspective-origin	perspectiveOrigin	Base 3D inicial da perspectiva Valores: x-axis y-axis, initial, inherit
position	position	Posicionamento do objeto. Valores: <u>static</u> , relative, absolute, fixed, inherit
quotes	quotes	Configura os elementos de marcação Valores: <u>none</u> , string string string string string, initial, inherit
resize	resize	Configura um element <div> como redimensionável Valores: none, both, horizontal, vertical, initial, inherit
right	right	Semelhante à propriedade bottom, porém, aplicado à direita.

		Valores: tamanho, porcentagem, <u>auto</u> , inherit
table-layout	tableLayout	Configura em fixo o tamanho das linhas e colunas da tabela Valores: automatic, fixed, initial, inherit
tab-size	tabSize	Configura o espaçamento do elemento <pre> Valores: number, length, initial, inherit
text-align	textAlign	Alinhamento do texto. Valores: left, right, center, justify, stringAlinhamento, inherit
text-align-last	textAlignLast	Alinha a última linha do parágrafo Valores: auto, left, right, center, justify, start, end, initial, inherit
text-decoration	textDecoration	Decoração do texto. Valores: <u>none</u> , underline, overline, line-through, blink, inherit
text-justify	textJustify	Configura o alinhamento do texto em justificado.
text-indent	textIndent	Indentação da primeira linha do texto. Valores: tamanho, porcentagem, inherit.
text-overflow	textOverflow	Configura o texto excedente ao container Valores: clip, ellipsis, string, initial, inherit
text-shadow	textShadow	Lista de sombras aplicadas ao texto. Valores: <u>none</u> , {cor, [distanciaX & distanciaY & desfoque]}, inherit
text-transform	textTransform	Define o fluxo de caracteres maiúsculos e minúsculos do texto. Valores: capitalize, uppercase, lowercase, <u>none</u> , inherit
top	top	Semelhante à propriedade bottom, porém, aplicado ao topo. Valores: tamanho, porcentagem, <u>auto</u> , inherit
transform	transform	Aplica uma transformação como rotação e escala por exemplo a uma <div> Valores: <u>none</u> , transform-functions, initial, inherit
transform-origin	transformOrigin	Ponto de origem, inicial para aplicar a transformação Valores: x-axis, y-axis, z-axis, initial, inherit
transform-style	transformStyle	Define ou retorna como elementos aninhados são renderizados no espaço 3D. Valores: flat, preserve-3d, initial, inherit
transition	transition	Define as configurações da transição. Valores: property duration timing-function delay, initial, inherit
transition-property	transitionProperty	Propriedade de configuração da transição Valores: <u>none</u> , all property, initial, inherit
transition-duration	transitionDuration	Tempo de duração da transição Valores: time, initial, inherit
transition-timing-function	transitionTimingFunction	Configura curva de velocidade da transição, velocidade do andamento Valores: ease linear, ease-in, ease-out, ease-in-out, cubic-bezier(), Initial, inherit
transition-delay	transitionDelay	Tempo para iniciar a transição Valores: time, initial, inherit
vertical-align	verticalAlign	Alinhamento vertical. Valores: <u>baseline</u> , sub, super, top, text-top, middle, bottom, text-bottom, porcentagem, tamanho, inherit.
visibility	visibility	Visibilidade do objeto. Valores: visible, hidden, collapse, <u>inherit</u>
word-break	wordBreak	Configura a regra de quebra de linha do texto Valores: normal, break-all, keep-all, initial, inherit
word-wrap	wordWrap	Configura se uma palavra poderá ser quebrada para próxima linha Valores: normal, break-word, initial, inherit
width	width	Largura do objeto. Valores: tamanho, porcentagem, <u>auto</u> , inherit
word-spacing	wordSpacing	Diatância entre as palavras. Valores: <u>normal</u> , tamanho, inherit
z-index	zIndex	Profundidade do objeto, empilhamento, camadas. Valores: <u>auto</u> , profundidade, inherit

## Formatação condicional

Uma das possibilidades de unir CSS e javascript é criar rotinas de formatação condicional, ou seja, formatar um elemento com um estilo visual dependendo de uma condição.

Vamos começar com um código simples, formatar um valor de vermelho se for menor que dez e azul se for maior ou igual a dez.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

      var num=5;

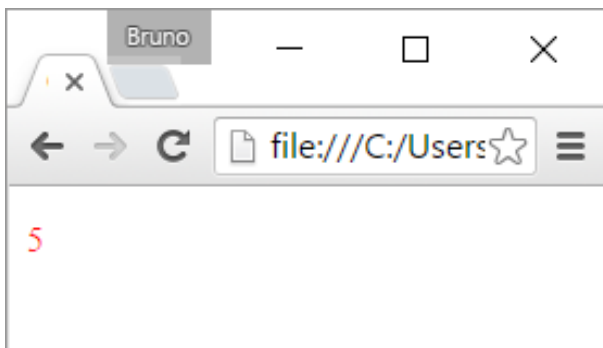
      document.write("<p id=tx>" + num + "</p>");
      var txt=document.getElementById("tx");

      if(txt.innerHTML < 10){
        txt.style.color="#F00";
      }else{
        txt.style.color="#00F";
      }

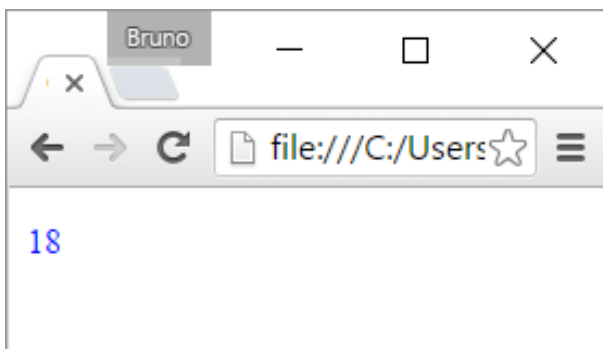
    </script>
  </head>
  <body>

    </body>
</html>
```

Para num com valor 5 o resultado do código é o numeral cinco formatado da cor vermelho, veja a ilustração a seguir.



Mudando o valor da variável para 18 a cor será azul.



Inicialmente criamos a variável num com valor 5.

```
var num=5;
```

Em seguida imprimimos este valor na tela dentro de uma tag <p> com id "tx".

```
document.write("<p id=tx>" + num + "</p>");
```

Para facilitar o uso no código associamos este elemento à variável "txt".

```
var txt=document.getElementById("tx");
```

O último passo é verificar o valor da <tag>, se for menor que 10 formata em vermelho e se for maior ou igual a 10 formata em azul.

```
if(txt.innerHTML < 10){
    txt.style.color="#F00";
}else{
    txt.style.color="#00F";
}
```

Pronto, viu como é simples? Vamos a outra situação.

Imagine uma tabela de estoque onde desejamos formatar em vermelho os produtos que tenham valor baixo no estoque, mas queremos que isto seja feito de forma automática?

Isto é possível com a formatação condicional, vamos ver como podemos implementar este recurso.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

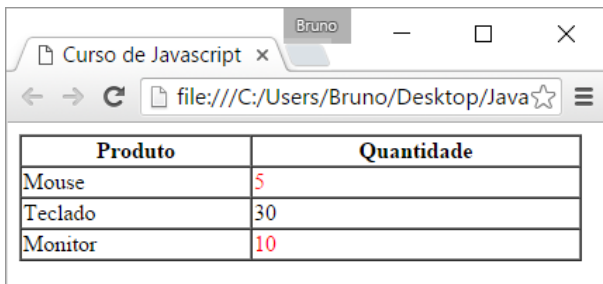
        function condicional() {
            var tds=document.getElementsByTagName("td");
            var i;
            for(i=0; i<tds.length; i++){
                if(tds[i].innerHTML < 15){
                    tds[i].style.color="#F00";
                }
            }
        }

    </script>
  </head>
  <body onload="condicional()">

    <table width="400" border="1" cellspacing="0">
      <tr>
        <th>Produto</th><th>Quantidade</th>
      </tr>
      <tr>
        <td>Mouse</td><td>5</td>
      </tr>
      <tr>
        <td>Teclado</td><td>30</td>
      </tr>
      <tr>
        <td>Monitor</td><td>10</td>
      </tr>
    </table>

  </body>
</html>
```

No código acima criamos uma tabela bem simples com três produtos, cada um com um valor relacionado à quantidade deste produto no estoque.



Produto	Quantidade
Mouse	5
Teclado	30
Monitor	10

Vamos formatar em vermelho os produtos que tiverem a quantidade menor que 10.

Vamos ao código da formatação.

O primeiro passo foi declarar o nome da função.

```
function condicional(){
```

Na sequência associamos todos as tags <td> à variável “tds”, como existem mais de uma, a variável “tds” na verdade é um array, o primeiro <td> fica na posição [0], o segundo <td> fica na posição [1] e assim por diante.

```
var tds=document.getElementsByTagName("td");
```

Criamos uma variável “i” para servir de índice para o FOR que iremos criar, afinal, precisamos percorrer o array com as tags <td>.

```
var i;
```

Aqui vai o loop FOR para percorrer o array “tds”.

```
for(i=0; i<tds.length; i++){
```

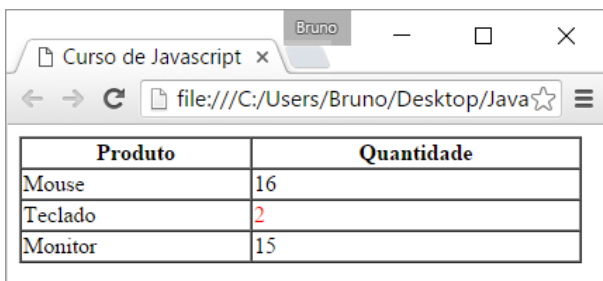
Para cada iteração do FOR, vamos testar uma posição do array “tds”, ou seja, o valor de uma tag <td>, comparando esse valor se é menor que 15, esse será o limite do nosso estoque.

```
if(tds[i].innerHTML < 15){
```

Se o valor for menor que 15, será formatado de vermelho.

```
tds[i].style.color="#F00";
```

Altere os valores relacionados à quantidade de cada um dos produtos e veja que serão automaticamente formatados.



Produto	Quantidade
Mouse	16
Teclado	2
Monitor	15

Em nosso código formatamos somente a cor do texto, você pode criar uma rotina para formatar a cor da célula, mudar o tamanho da fonte, etc.



## Função para cálculo de IMC

Vamos praticar um pouco o uso de funções criando um função que faz o cálculo do IMC (Índice de Massa Corporal), vamos obter os valores digitados nos campos de texto, iremos calcular emitir uma caixa de mensagem alert com o resultado.

Veja o código completo a seguir.

```

<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      function imc(){
        var valt_tmp=document.f_imc.f_alt.value;
        var vpes_tmp=document.f_imc.f_pes.value;

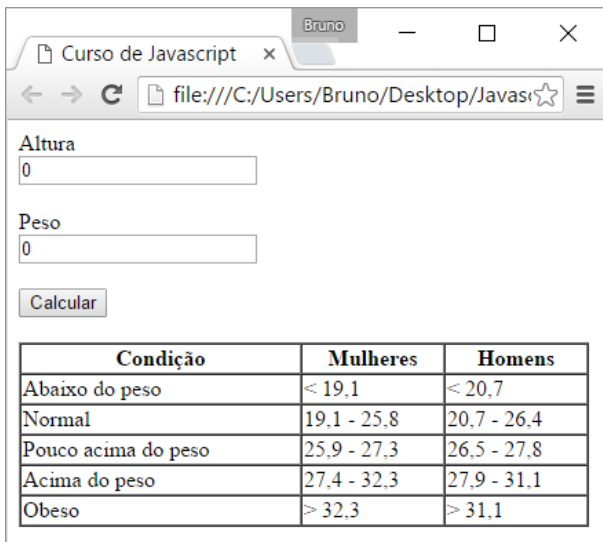
        if((valt_tmp == 0)||(vpes_tmp == 0)){
          alert("Digite o peso ou altura");
        }else{
          var valt=valt_tmp.replace(",",".");
          var vpes=vpes_tmp.replace(",",".");

          var vimc=(vpes/Math.pow(valt,2)).toFixed(1);
          alert("O IMC é: " + vimc);
        }
      }
    </script>
  </head>
  <body>

    <form name="f_imc">
      <label>Altura</label><br>
      <input type="text" name="f_alt" value="0"><br><br>
      <label>Peso</label><br>
      <input type="text" name="f_pes" value="0"><br><br>
      <input type="button" value="Calcular" onclick="imc()">
    </form>
    <br>
    <table border="1" cellspacing="0">
      <tr>
        <th width="200">Condição</th><th width="100">Mulheres</th><th width="100">Homens</th>
      </tr>
      <tr>
        <td>Abaixo do peso</td><td> < 19,1 </td><td> < 20,7 </td>
      </tr>
      <tr>
        <td>Normal</td><td> 19,1 - 25,8 </td><td> 20,7 - 26,4 </td>
      </tr>
      <tr>
        <td>Pouco acima do peso</td><td> 25,9 - 27,3 </td><td> 26,5 - 27,8 </td>
      </tr>
      <tr>
        <td>Acima do peso</td><td> 27,4 - 32,3 </td><td> 27,9 - 31,1 </td>
      </tr>
      <tr>
        <td>Obeso</td><td> > 32,3 </td><td> > 31,1 </td>
      </tr>
    </table>

  </body>
</html>

```



Condição	Mulheres	Homens
Abaixo do peso	< 19,1	< 20,7
Normal	19,1 - 25,8	20,7 - 26,4
Pouco acima do peso	25,9 - 27,3	26,5 - 27,8
Acima do peso	27,4 - 32,3	27,9 - 31,1
Obeso	> 32,3	> 31,1

Vou explicar a função que criamos.

Primeiramente informamos o nome da função na declaração.

```
function imc(){
```

Em seguida criamos as variáveis com os valores dos campos digitados no formulário.

```
var valt_tmp=document.f_imc.f_alt.value;
var vpes_tmp=document.f_imc.f_pes.value;
```

Após obter os valores precisamos verificar se foram digitados o peso e a altura, se os valores forem zero, significa que não foram digitados valores, então não podemos realizar o cálculo.

```
if((valt_tmp == 0) || (vpes_tmp == 0)){
    alert("Digite o peso ou altura");
}else{
    var valt=valt_tmp.replace(",",".");
    var vpes=vpes_tmp.replace(",",".");

    var vimc=(vpes/Math.pow(valt,2)).toFixed(1);

    alert("O IMC é: " + vimc);
}
```

Na função acima usamos dois métodos novos “replace” e “toFixed”, vou falar sobre eles.

## replace()

O método “replace()” é bem simples, basicamente faz a troca/substituição/replace de um caractere por outro. O método possui uma sintaxe simplificada, veja.

```
var v2 = v1.replace("c1", "c2");
```

onde:

v2 → Variável com o novo texto após a troca do(s) caractere(s).

v1 → Variável com o texto original antes da substituição do(s) caractere(s).

c1 → Caractere(s) a ser substituído.

c2 → Caractere que irá substituir.

No nosso código do IMC usamos o seguinte comando.

```
var vpes=vpes_tmp.replace(",",".");
```

A variável vpes vai receber o novo texto, o original está na variável vpes\_tmp, serão substituídos todos os caracteres vírgula por ponto.

Usamos esta função no cálculo do IMC porque a pessoa pode digitar vírgula ao invés de ponto e a vírgula não é o separador numérico para casas decimais e sim o ponto.

### *toFixed()*

O uso do método “toFixed()” foi empregado para que o valor fosse arredondado com um número específico de casas decimais, veja a sintaxe do método.

```
var v2=v1.toFixed(casas);
```

onde:

v2 → Variável que receberá o novo numeral com as casas decimais especificadas.

v1 → Variável que possui algum numeral com várias casas decimais.

casas → Valor que indicará a quantidade de casas decimais desejadas.

Em nosso programa usamos o seguinte código.

```
var vimc=(vpes/Math.pow(val,2)).toFixed(1);
```

Neste caso, a variável vimc está recebendo o cálculo do IMC retornado com somente uma casa decimal.

## *Slider simples*

Vamos aprender como criar um slider simples, com três imagens e dois botões para mudar os slides, anterior e próximo, outro recurso interessante será parar a troca dos slides enquanto o mouse estiver sobre um slide.

Vamos criar uma <div> que vai ser a base do nosso slider, nosso script vai simplesmente aplicar as imagens dos slides como backgroundImage da <div> e irá mudar a imagem de três em três segundos.

Veja os códigos HTML e CSS para criação da <div> dos botões para avançar e retroceder os slides.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">

    <style>
      #dvSlider{
        width:500px;
        height:150px;
      }
      #bt{
        color:#FFF;
        background-color:#000;
        border:0px;
      }
    </style>

    <script>
```

```

    </script>
</head>
<body onload="">

    <div id="dvSlider">
        <input type="button" value="<" id="bt" onclick="">
        <input type="button" value=">" id="bt" onclick="">
    </div>

</body>
</html>

```

Agora vamos implementar os scripts para tudo funcionar.

Primeiramente vamos adicionar três variáveis e um array. O array irá armazenar os nomes dos arquivos de imagem, que são os slides.

```
var slides=new Array("s1.jpg","s2.jpg","s3.jpg");
```

Precisamos armazenar o tamanho do array para saber quantos slides temos disponíveis.

```
var tam=slides.length;
```

Uma variável para indicar o slide atual e uma para receber o timer para troca automática dos slides.

```
var satual=1;
var tmpSlider;
```

Em seguida vamos criar a função responsável pela troca automática dos slides. Esta função irá incrementar a variável “satual” e irá verificar que o valor do slide atual é maior ou igual à quantidade de slides, se for volta o valor para “0”, pois, é a primeira posição do array com os nomes dos slides. Por fim simplesmente aplica a imagem do backgroundImage da <div> por formatação CSS.

```

function trocaAutomatica(){
    satual++;
    if(satual >= tam){
        satual=0;
    }
    document.getElementById("dvSlider").style.backgroundImage="url('"+slides[satual]+"')";
}

```

Precisamos de uma função para iniciar o trabalho de troca automática dos slides, então vamos criar a função para esta tarefa. A função vai configurar a imagem de fundo da <div> de acordo com o índice “satual” no array de imagens e iniciar o intervalo (timer) para troca das imagens de três em três segundos.

```

function iniciaSlider(){
    document.getElementById("dvSlider").style.backgroundImage="url('"+slides[satual]+"')";
    tmpSlider=setInterval(trocaAutomatica,3000);
}

```

Precisamos implementar a função que irá parar a troca de slides quando o mouse estiver sobre um slide. É uma função simples, somente com um comando clearInterval.

```

function parar(){
    clearInterval(tmpSlider);
}

```

Por último implementamos a função que fará a troca das imagens de acordo com os botões, anterior ou próximo. A função usa um parâmetro “dir” para saber se precisa avançar ou retroceder a imagem

```
function troca(dir){
    satual+=dir;
    if(satual < 0){
        satual=2;
    }else if(satual >= tam){
        satual=0;
    }
    document.getElementById("dvSlider").style.backgroundImage="url('"+slides[satual]+"')";
    clearInterval(tmpSlider);
    tmpSlider=setInterval(trocaAutomatica,3000);
}
```

Pronto, com todo script criado, precisamos associar as funções aos eventos devidos no <body> e na <div>, faça estas modificações conforme os exemplos a seguir.

<body onload="iniciaSlider()">

<div id="dvSlider" onmouseover="parar()" onmouseout="iniciaSlider()">

Confira a seguir o código completo e uma imagem de exemplo do slider.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">

    <style>
      #dvSlider{
        width:500px;
        height:150px;
      }
      #bt{
        color:#FFF;
        background-color:#000;
        border:0px;
      }
    </style>

    <script>

      var slides=new Array("s1.jpg","s2.jpg","s3.jpg");
      var tam=slides.length;
      var satual=1;
      var tmpSlider;

      function trocaAutomatica(){
        satual++;
        if(satual >= tam){
          satual=0;
        }
        document.getElementById("dvSlider").style.backgroundImage="url('"+slides[satual]+"')";
      }

      function iniciaSlider(){
        document.getElementById("dvSlider").style.backgroundImage="url('"+slides[satual]+"')";
        tmpSlider=setInterval(trocaAutomatica,3000);
      }

      function parar(){
        clearInterval(tmpSlider);
      }

      function troca(dir){
```

```

        satual+=dir;
        if(satual < 0){
            satual=2;
        }else if(satual >= tam){
            satual=0;
        }
        document.getElementById("dvSlider").style.backgroundImage="url('"+slides[satual]+"')";
        clearInterval(tmpSlider);
        tmpSlider=setInterval(trocaAutomatica,3000);
    }

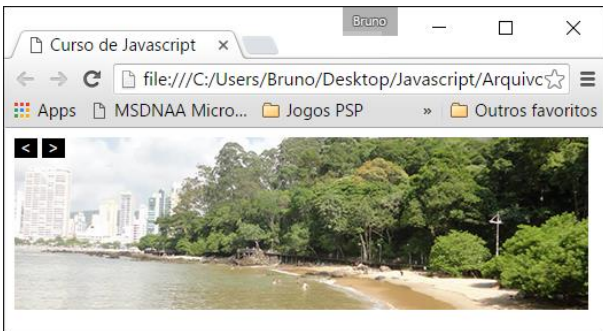
</script>
</head>
<body onload="iniciaSlider()">

    <div id="dvSlider" onmouseover="parar()" onmouseout="iniciaSlider()">
        <input type="button" value="<" id="bt" onclick="troca(-1)">
        <input type="button" value=">" id="bt" onclick="troca(1)">
    </div>

</body>
</html>

```

OBS: Como a <div> foi configurada com tamanho 500 x 150, as imagens devem seguir este tamanho, se for usar imagens maiores ou menores aumente ou diminua o tamanho da div conforme o tamanhos das imagens.



## Adicionando e ou removendo atributos HTML via javascript

É muito comum precisarmos adicionar ou remover atributos HTML em algumas tags via javascript, felizmente temos métodos que fazem isso com muita facilidade, são os métodos `setAttribute()` e `removeAttribute()`.

Vamos entender como utilizar estes métodos.

### *setAttribute()*

A utilização deste método é bem simples, basta informar dois parâmetros de entrada (strings), o primeiro referente ao nome do atributo HTML que deseja adicionar à tag e o segundo referente ao valor do atributo.

Veja um exemplo a seguir onde adicionamos a classe CSS "txth1" na tag que tem id="cfb".

```
document.getElementById("cfb").setAttribute("class","txth1");
```

Vamos ver um código de exemplo mais completo.

```

<!doctype html>
<html lang="pt-br">
    <head>
        <title>Curso de Javascript</title>
        <meta charset="UTF-8">
        <style>
            .txth1{
                color:#F00;
                font-size:45pt;
            }
        </style>

```

```

<script>
    function classe(){
        document.getElementById("cfb").setAttribute("class","txth1");
        document.getElementById("cfb").setAttribute("style","background-color:#AAA;");
    }
</script>
</head>
<body onload="classe()">

    <h1 id="cfb">Canal Fessor Bruno</h1>

</body>
</html>

```

Veja no exemplo acima que temos uma tag <h1> no body, note que esta tag tem o id="cfb".

No script temos uma função com nome classe, vamos entender os comandos desta função.

O primeiro comando configura o atributo "class" com o valor "txth1", ou seja, adiciona a classe "txth1" à tag <h1> que tem id igual a "cfb".

```
document.getElementById("cfb").setAttribute("class","txth1");
```

O segundo comando adiciona o atributo "style" com valor "background-color:#AAA;", ou seja, configura a cor de fundo para cinza.

```
document.getElementById("cfb").setAttribute("style","background-color:#AAA;");
```

Para chamar esta função e aplicar os atributos associamos a função ao evento "onload" do <body>, assim quando a página for carregada a função será chamada e os atributos adicionados.

Veja a seguir o resultado final deste código.



O método setAttribute() adiciona o atributo indicado caso ele não exista na tag, se o atributo já existir ele simplesmente será alterado, não será adicionar outro atributo com o mesmo nome.

## *removeAttribute()*

De forma contrária ao conteúdo anterior, nós podemos remover atributos das tags html, o processo é bem simples, veja um exemplo de comando para remover o atributo "style" da tag com id "cfb".

```
document.getElementById("cfb").removeAttribute("style");
```

Vamos adicionar este comando ao nosso script anterior e ver o resultado final.

```

<script>
    function classe(){
        document.getElementById("cfb").setAttribute("class","txth1");
        document.getElementById("cfb").setAttribute("style","background-color:#AAA;");
        document.getElementById("cfb").setAttribute("style","background-color:#000;");
        document.getElementById("cfb").removeAttribute("style");
    }

```

```
</script>
```

Desta maneira adicionamos a atributo style, alteramos seu valor e removemos ao final, então veja o resultado final do script com esta alteração.



Note que o texto <h1> não tem mais a cor de fundo configurada.

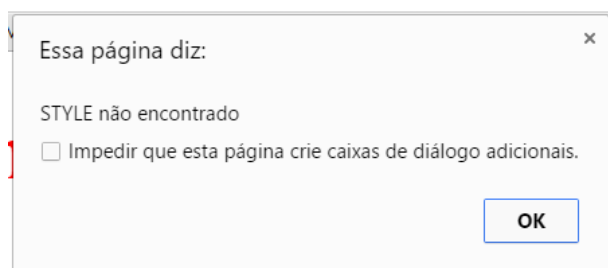
### *hasAttribute()*

Outro comando que podemos usar para controlar os atributos nas tags htmls é o comando “hasAttribute()”, este método simplesmente verifica se a tag já possui o atributo indicado, se possuir o método retorna “true” se não possuir o método retorna “false”.

Vamos implementar este método no script anterior.

```
<script>
function classe(){
    document.getElementById("cfb").setAttribute("class","txth1");
    document.getElementById("cfb").setAttribute("style","background-color:#AAA;");
    document.getElementById("cfb").setAttribute("style","background-color:#000;");
    document.getElementById("cfb").removeAttribute("style");
    if(document.getElementById("cfb").hasAttribute("style")){
        alert("a A tag possui o atributo STYLE");
    }else{
        alert("STYLE não encontrado");
    }
}
</script>
```

Ao atualizar a página o resultado será a caixa alert com a mensagem “STYLE não encontrado”.



## *String – Funções para manipular strings*

Existem várias funções disponíveis para manipular strings, conhecer estas funções é muito importante, embora pareça insignificante, saber manipular strings nos abre uma série de possibilidades.

Neste capítulo vamos aprender diversas funções para manipular strings, faça muito proveito e use sua criatividade para ver as possibilidades de cada método.



## match()

O método match procura um texto especificado em uma string, o método pode retornar o valor procurado como único valor de retorno, um array com todas as ocorrências encontradas ou “null” caso nada seja encontrado.

A sintaxe é simples, veja a seguir.

```
string.match(procura);
```

Vamos a alguns exemplos.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Curso de Javascript</title>
    <script>

      var texto="Curso de Javascript - Canal Fessor Bruno";
      var resultado=texto.match("Canal");

      document.write(resultado);

    </script>
  </head>
  <body>

  </body>
</html>
```

No código acima criamos uma variável com um texto, em seguida criamos outra variável que vai receber o resultado da busca do método match que está procurando o texto “Canal”.

O resultado deste código é a impressão do texto “Canal” na tela, um detalhe importante é que se for pesquisado o texto “canal”, com “c” minúsculo o resultado será “null”, pois, o padrão é a pesquisa exata.

Quando pesquisamos usando aspas, independentemente do número de ocorrências encontradas a quantidade informada será sempre 1, ou seja, se tivéssemos 5 palavras “Canal”, seria retornado a palavra “Canal” e somente uma ocorrência. Veja a alteração a seguir em vermelho mostrando isso.

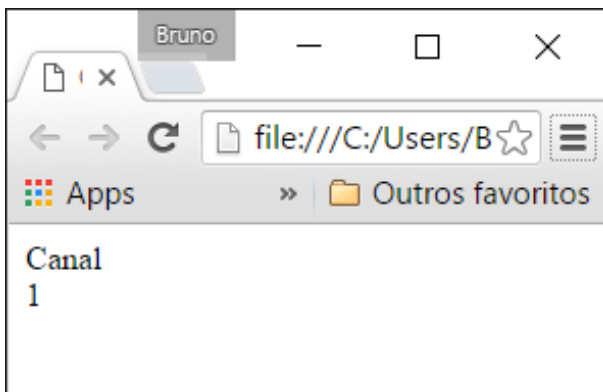
```
<script>

  var texto="Canal Canal Canal Canal Canal";
  var resultado=texto.match("Canal");

  document.write(resultado + "<br>");
  document.write(resultado.length);

</script>
```

O resultado deste código.



Imprimimos o resultado da palavra encontrada e a quantidade de ocorrências encontradas, mas note que está mostrando só uma ocorrência e temos cinco palavras “Canal”.

Mesmo se informamos o resultado em forma de vetor o resultado será o mesmo, veja.

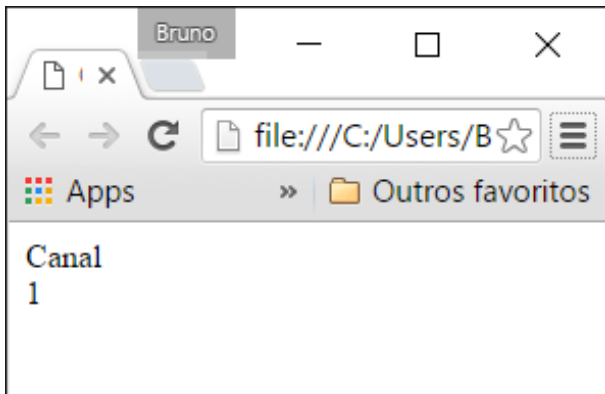
```
<script>

    var texto="Canal Canal Canal Canal Canal";
    var resultado=texto.match("Canal");

    document.write(resultado[0] + "<br>");
    document.write(resultado.length);

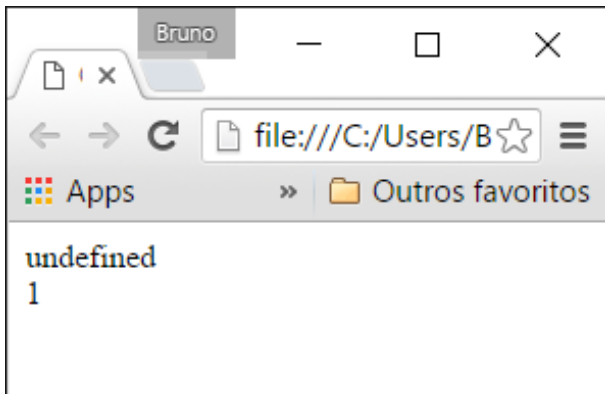
</script>
```

Veja que o resultado é o mesmo.



Se tentar mudar o índice para 1, verá que não existem mais itens no array.

```
document.write(resultado[1] + "<br>");
```



Como melhorar isto e passar a armazenar os resultados em um array? É simples, basta mudar a forma de passar o parâmetro e usar um modificador, veja.

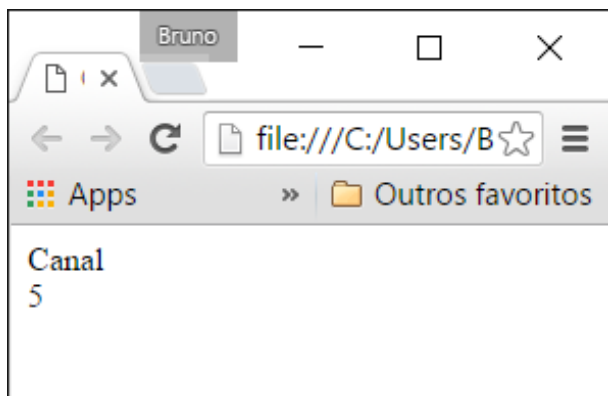
```
<script>

    var texto="Canal Canal Canal Canal Canal";
    var resultado=texto.match(/Canal/g);

    document.write(resultado[0] + "<br>");
    document.write(resultado.length);

</script>
```

Trocamos aspas por barra e usamos o modificador “g”, desta maneira as ocorrências são armazenadas uma em cada posição do vetor, veja o resultado desta alteração.



O modificador “g” diz ao método para encontrar todas as ocorrências da palavra e não parar na primeira encontrada, podemos ainda usar os modificadores “i” e “m”, veja a tabela a seguir.

Modificador	Descrição
i	Busca sem case-sensitive, ou seja, não diferencia letras maiúsculas de minúsculas
g	Diz ao método para encontrar todas as ocorrências da palavra e não parar na primeira encontrada, cada ocorrência é armazenada em uma posição do array
m	Pesquisa normal sem armazenar em forma de array

Agora sabemos como pesquisar ignorando caracteres maiúsculos de minúsculos, veja a alteração.

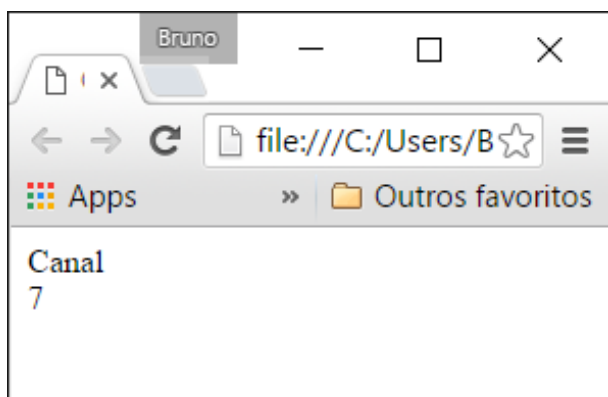
```
<script>

var texto="Canal cAnal caNal canAl canaL canal CANAL";
var resultado=texto.match(/canal/gi);

document.write(resultado[0] + "<br>");
document.write(resultado.length);

</script>
```

Note que agora usamos dois modificadores “g” e “i”, então confira a seguir o resultado deste código.



Vamos a mais uma pequena alteração em nosso código tirando o índice do array.

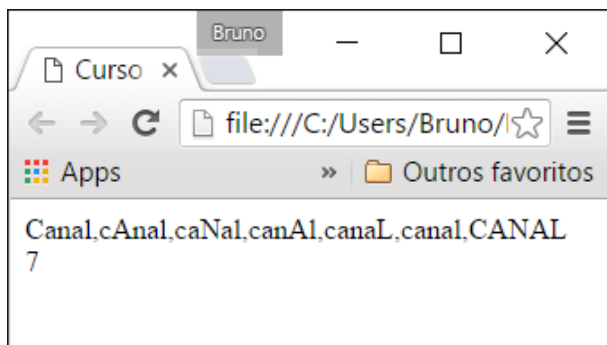
```
<script>

var texto="Canal cAnal caNal canAl canaL canal CANAL";
var resultado=texto.match(/canal/gi);

document.write(resultado + "<br>");
document.write(resultado.length);

</script>
```

Veja que agora são impressos todas as ocorrências de uma só vez.



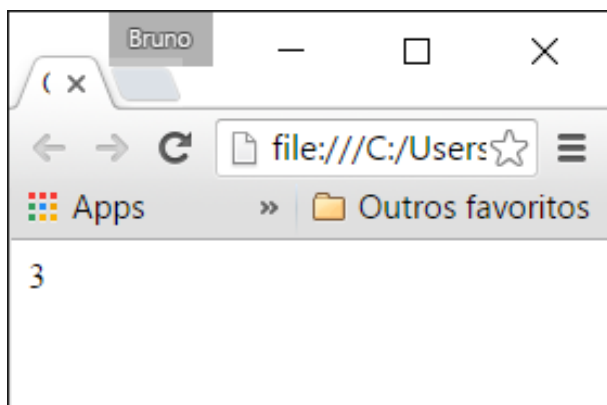
Quer procurar por letras? O processo é o mesmo, veja a alteração do código onde procuramos por ocorrências da letra “o” e imprimimos a quantidade de letras “o” encontradas.

```
<script>

var texto="Canal Fessor Bruno - Curso de Javascript - CFB";
var resultado=texto.match(/o/gi);

document.write(resultado.length);

</script>
```



Ainda podemos melhorar nossa pesquisa usando colchetes, por exemplo, se for preciso pesquisar todas as letras “a” e “o”? É simples.

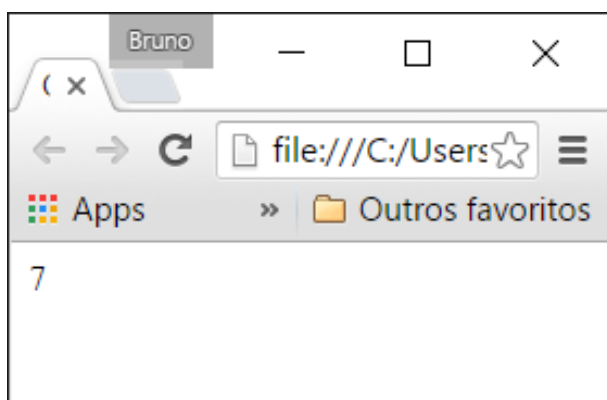
```
<script>

var texto="Canal Fessor Bruno - Curso de Javascript - CFB";
var resultado=texto.match(/[oa]/gi);

document.write(resultado.length);

</script>
```

Agora veja o novo resultado.



O resultado sete é relativo ao número de letras “a” somado ao número de letras “o”.

Confira a seguir uma tabela com outras opções de pesquisa usando colchetes.

Expressão	Descrição
[abcd]	Pesquisa pelos caracteres “a”, “b”, “c” e “d”
[^ab]	Pesquisa por todos os caracteres, menos os “a” e “b”.
[a-f]	Pesquisa pelos caracteres de “a” até “f”
[^a-f]	Pesquisa por todos os caracteres, menos os de “a” até “f”
[a f]	Pesquisa pelos caracteres “a” e “f”, semelhante à primeira opção.

Outras opções que podemos para implementar nossa pesquisa são os “metacaracteres”, por exemplo, se precisarmos saber quantos espaços existem na string, basta usar o “metacaractere” espaço, veja o exemplo a seguir.

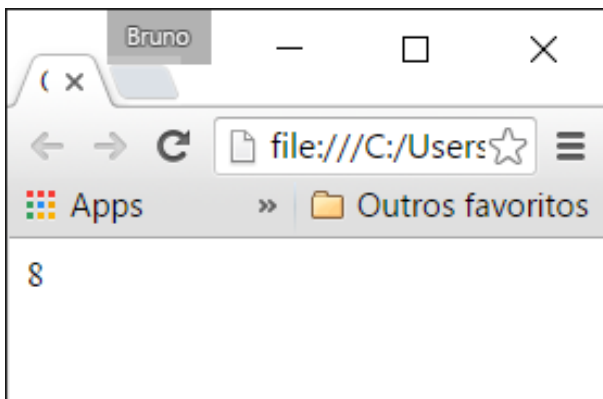
```
<script>

var texto="Canal Fessor Bruno - Curso de Javascript - CFB";
var resultado=texto.match(/\s/g);

document.write(resultado.length);

</script>
```

Assim obtemos a contagem dos espaços, veja o resultado.



Existem vários “metacaracteres” que podemos usar, confira a tabela a seguir.

Metacaratere	Descrição
\w	Pesquisa somente por letras e numerais, ignorando espaços, traços, etc.
\W	Pesquisa por caracteres diferentes de números e letras
\d	Pesquisa por numerais
\D	Pesquisa por todos os caracteres que não são numerais
\s	Pesquisa pelos espaços
\S	Pesquisa por todos os caracteres, menos os espaços
\b	Pesquisa por ocorrência que iniciem ou terminem com uma letra ou número, cada ocorrência que iniciar e terminar com um número ou letra conta como 2
\B	Pesquisa por ocorrência que NÃO iniciem ou terminem com uma letra ou número
\O	Procura por caracteres nulos.
\n	Procura por quebra de linha.
\r	Procura por caractere de “retorno de carro” ENTER.
\t	Procura por caractere de tabulação TAB
\v	Procura por caractere de tabulação vertical
s+	Procura por palavras que contenham uma ou mais letras “s”

Vamos aprender na sequência outros métodos para trabalhar com strings.

## search()

O método `search()` procura pela ocorrência e retorna sua posição dentro da string, a posição é em relação ao primeiro caractere da ocorrência, veja o código.

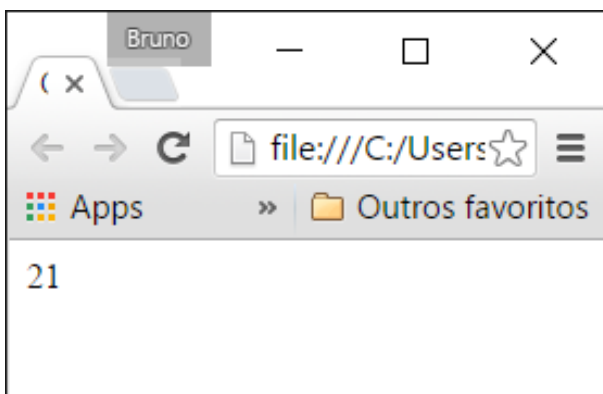
```
<!DOCTYPE html>
<html>
  <head>
    <title>Curso de Javascript</title>
    <script>

      var texto="Canal Fessor Bruno - Curso de Javascript - CFB";
      var resultado=texto.search(/Curso/);

      document.write(resultado);

    </script>
  </head>
  <body>
  </body>
</html>
```

O método “search” irá procurar pela primeira ocorrência da palavra “Curso” com “C” maiúsculo, veja o resultado deste código.



Foi retornada a posição da primeira letra da ocorrência encontrada.

`var texto="Canal Fessor Bruno - Curso de Javascript - CFB";`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	...
C	a	n	a	l		F	e	s	s	o	r		B	r	u	n	o		-		C	u	r	s	o	

Note na ilustração de exemplo acima que a palavra “Curso” começa na posição 21.

**OBS:** Os modificadores que aprendemos anteriormente também podem ser usados no método “search” da mesma maneira que no método “match”.

## replace()

Este método substitui uma string por outra, simples assim, ele pesquisa a string informada, como no método “match” e a substitui por outra string nas aspas informada como segundo parâmetro, veja o código de exemplo a seguir que troca o texto “CFB” pelo texto “www.cfbcursos.com.br”.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Curso de Javascript</title>
    <script>
```

```

        var texto="Canal Fessor Bruno - Curso de Javascript - CFB";
        var resultado=texto.replace(/cfb/i,"www.cfbcursos.com.br");

        document.write(resultado);

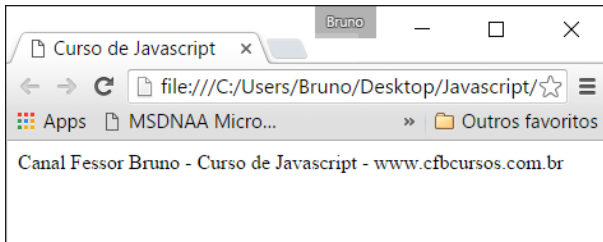
    </script>
</head>

    <body>

    </body>
</html>

```

O resultado é o mostrado a seguir.



Note pelo código que o texto “Canal Fessor Bruno - Curso de Javascript – CFB” teve a string “CFB” trocada e foi impresso o texto “Canal Fessor Bruno - Curso de Javascript – www.cfbcursos.com.br”

## charAt()

Retorna o caractere na posição indicada como parâmetro do método.

```

<script>

    var texto="Canal Fessor Bruno";
    var resultado=texto.charAt(6);

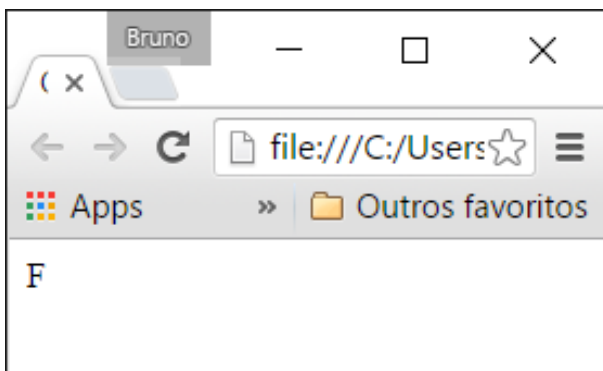
    document.write(resultado);

</script>

```

No código acima o método retorna a letra “F”, pois é o caractere que está na posição 6.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C	a	n	a	l		F	e	s	s	o	r		B	r	u	n	o



## charCodeAt()

Retorna o código do caractere na posição indicada como parâmetro do método.

```

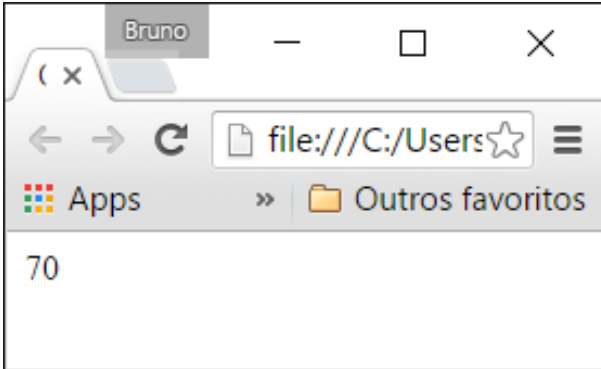
<script>

    var texto="Canal Fessor Bruno";

```

```
var resultado=texto.charCodeAt(6);  
  
document.write(resultado);  
  
</script>
```

No código acima o método retorna o código a letra “F” que é o código 70, pois é o caractere que está na posição 6.

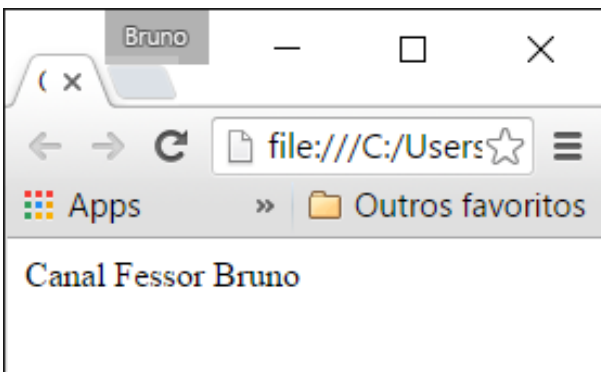


### *concat()*

Concatena, funde, une, junta uma string em outra, no código a seguir juntamos as strings txt1 e txt2.

```
<script>  
  
var txt1="Canal ";  
var txt2="Fessor Bruno";  
var resultado=txt1.concat(txt2);  
  
document.write(resultado);  
  
</script>
```

Como resultado temos.



### *fromCharCode()*

Converte um código informado no caractere correspondente, o programa a seguir converte o código 66 no caractere correspondente que é o “B”.

```
<script>  
  
var resultado=String.fromCharCode(66);  
  
document.write(resultado);  
  
</script>
```



## indexOf()

Retorna a posição do primeiro caractere especificado a ser encontrado, no exemplo procuramos a posição da letra “B” e será retornada a posição 13.

```
<script>

  var texto="Canal Fessor Bruno";
  var resultado=texto.indexOf("B");

  document.write(resultado);

</script>
```

## lastIndexOf()

Retorna a posição do último caractere especificado a ser encontrado, no exemplo procuramos a posição da última letra “a” e será retornada a posição 13.

```
<script>

  var texto="Canal Fessor Bruno";
  var resultado=texto.lastIndexOf("a");

  document.write(resultado);

</script>
```

O resultado deste script é o valor 3 que é a posição da última letra “a” já que a primeira está na posição “1”.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C	a	n	a	l		F	e	s	s	o	r		B	r	u	n	o

## localeCompare()

Este método compara duas strings, se forem iguais retorna “0” zero, se a primeira for menor que a segunda retorna “-1” e se for maior, retorna “1”, então qualquer valor diferente de zero significa que as strings não são iguais. Em nosso código de exemplo o valor impresso será “0” zero, pois, as strings são idênticas.

```
<script>

  var texto1="Curso";
  var texto2="Curso";
  var resultado=texto1.localeCompare(texto2);

  document.write(resultado);

</script>
```

## slice()

O método slice() corta uma string de um ponto A até um ponto B especificados e retorna o valor recortado, nosso script a seguir retorna o texto “Bruno”.

```
<script>

  var texto="Canal Fessor Bruno";
  var resultado=texto.slice(13,18);

  document.write(resultado);

</script>
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
C	a	n	a	l		F	e	s	s	o	r		B	r	u	n	o	

Observe que foi recortado da posição 13 até a 18 e retornado o texto “Bruno”.

Podemos usar o método `slice()` com somente um parâmetro, desta maneira como no exemplo a seguir.

```
var resultado=texto.slice(6);
```

Desta maneira todo conteúdo antes da posição 6 será eliminado e será retornado o texto “Fessor Bruno”.

Podemos usar o método `slice()` em vetores também,

```
var transportes = ["avião ", "moto", "carro", "trem", "navio"];  
var terrestres = transportes.slice(1, 4);
```

Serão armazenado no array `terrestres` os transportes moto, carro e trem.

## *split()*

O método `split()` subdivide uma string sempre que encontrar o caractere especificado como divisor, no nosso código definimos o caractere divisor sendo o espaço, desta maneira será criado um array e em cada posição do array será inserida uma ocorrência.

```
<script>  
  
    var texto="Canal Fessor Bruno";  
    var resultado=texto.split(" ");  
  
    document.write(resultado.length);  
  
</script>
```

O resultado deste script é três, pois temos uma string com três palavras separadas com espaços uma da outra, então a palavra “Canal” fica na posição [0], “Fessor” fica na posição [1] e “Bruno” fica na posição [2].

Caso seja necessário criar um array com as letras separadas basta usar como o exemplo a seguir.

```
<script>  
  
    var texto="Canal Fessor Bruno";  
    var resultado=texto.split("");  
  
    document.write(resultado.length);  
  
</script>
```

Neste código o valor retornado como tamanho do array é treze, pois, agora cada caractere será armazenado em uma posição do array.

Ainda podemos indicar um limite para a divisão, veja o código a seguir que especifica um limite de 2, ou seja será feito o `split`, porém somente até a segunda ocorrência, então o código a seguir retorna “Canal,Fessor”.

```
<script>  
  
    var texto="Canal Fessor Bruno";  
    var resultado=texto.split(" ",2);  
  
    document.write(resultado);  
  
</script>
```

## *substr()*

O método `substr()` funciona de forma semelhante ao método `slice()`, com uma diferença básica no segundo parâmetro, que indica o tamanho do corte, veja o exemplo que retorna o texto “Bru”, indicamos o início do corte na posição 13 e o tamanho como 3, então, a partir da posição 13 serão recortados os 3 caracteres.

```
<script>
```

```
var texto="Canal Fessor Bruno";
var resultado=texto.substr(13,3);

document.write(resultado);

</script>
```

## *toLowerCase()*

Converte a string inteira para caracteres minúsculos, veja o código de exemplo.

```
<script>

var texto="Canal Fessor Bruno";
var resultado=texto.toLowerCase();

document.write(resultado);

</script>
```

O resultado do código será o texto todo com letras minúsculas.

## *toUpperCase()*

Converte a string inteira para caracteres maiúsculos, veja o código de exemplo.

```
<script>

var texto="Canal Fessor Bruno";
var resultado=texto.toUpperCase();

document.write(resultado);

</script>
```

O resultado do código será o texto todo com letras maiúsculas.

## *toString()*

Converte um valor qualquer em uma string.

```
<script>

var num=10;
var resultado=num.toString();

document.write(resultado);

</script>
```

Será impresso o numeral 10.

Uma forma de uso muito interessante do método `toString` é a possibilidade de retornar o numeral indicado na base, 2, base 8 ou base 16, ou seja, converter em binário, octal ou hexadecimal, veja o código de exemplo.

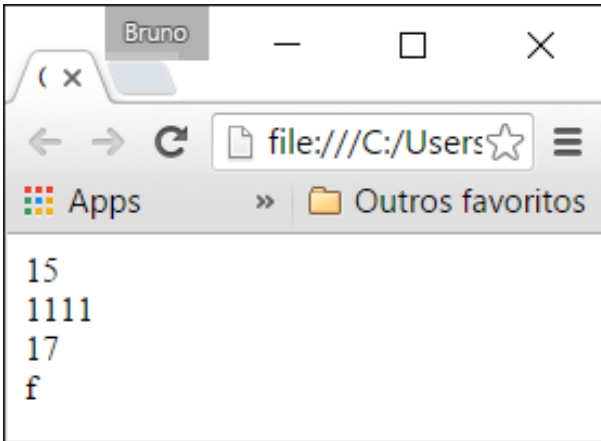
```
<script>

var decimal=15;
var binario=decimal.toString(2);
var octal=decimal.toString(8);
var hexadecimal=decimal.toString(16);

document.write(decimal + "<br>");
document.write(binario + "<br>");
document.write(octal + "<br>");
document.write(hexadecimal);

</script>
```

O código acima converte o numeral 15 para as bases, 2, 8 e 16, veja o resultado.



### *trim()*

Remove os espaços antes e depois da string especificada. No código abaixo serão removidos todos os espaços no início e no final da string, os espaços no meio da string não são removidos.

```
<script>

    var texto="    Canal    Fessor    Bruno    ";
    var resultado=texto.trim();

    alert(resultado);

</script>
```

### *Usando caracteres especiais na string*

Em alguns momentos vamos precisar usar aspas, adicionar tabulação, quebra de linha, etc, em nossas strings e para estes casos temos que indicar estes caracteres dentro da string, veja um exemplo a seguir de uma string com aspas.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Curso de Javascript</title>
        <script>

            var texto="Canal Fessor Bruno - \"CFB\" - Curso de Javascript";

            document.write(texto);

        </script>
    </head>

    <body>

    </body>
</html>
```

Como as aspas são os delimitadores da string, precisamos informar ao sistema quando as aspas não estão delimitando a string e sim estão ali para serem impressas.

Note que foi usada uma barra invertida antes das aspas que serão impressas, esta é a maneira de informar que estas aspas serão impressas.

A mesma técnica vale para os apóstrofes e para as barras invertidas, veja os exemplos.

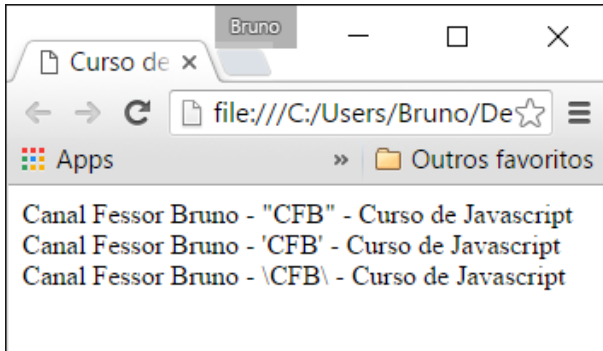
```
<script>

    var texto1="Canal Fessor Bruno - \"CFB\" - Curso de Javascript";
    var texto2="Canal Fessor Bruno - \'CFB\' - Curso de Javascript";
    var texto3="Canal Fessor Bruno - \\CFB\\ - Curso de Javascript";

    document.write(texto1 + "<br>" + texto2 + "<br>" + texto3);
```

```
</script>
```

E observe as impressões.



Outro caractere especial é o “\n” que insere uma quebra de linha na string, veja o código de exemplo a seguir.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Curso de Javascript</title>
    <script>

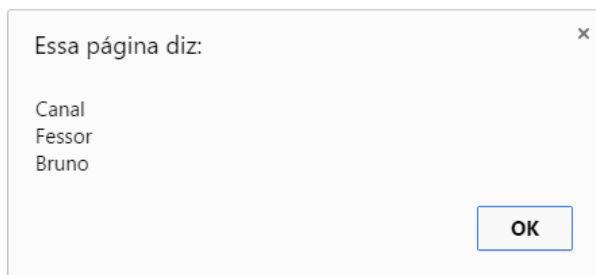
      var texto="Canal\nFessor\nBruno";

      alert(texto);

    </script>
  </head>
  <body>

  </body>
</html>
```

Note que foi usado um alert e não uma impressão na tela.



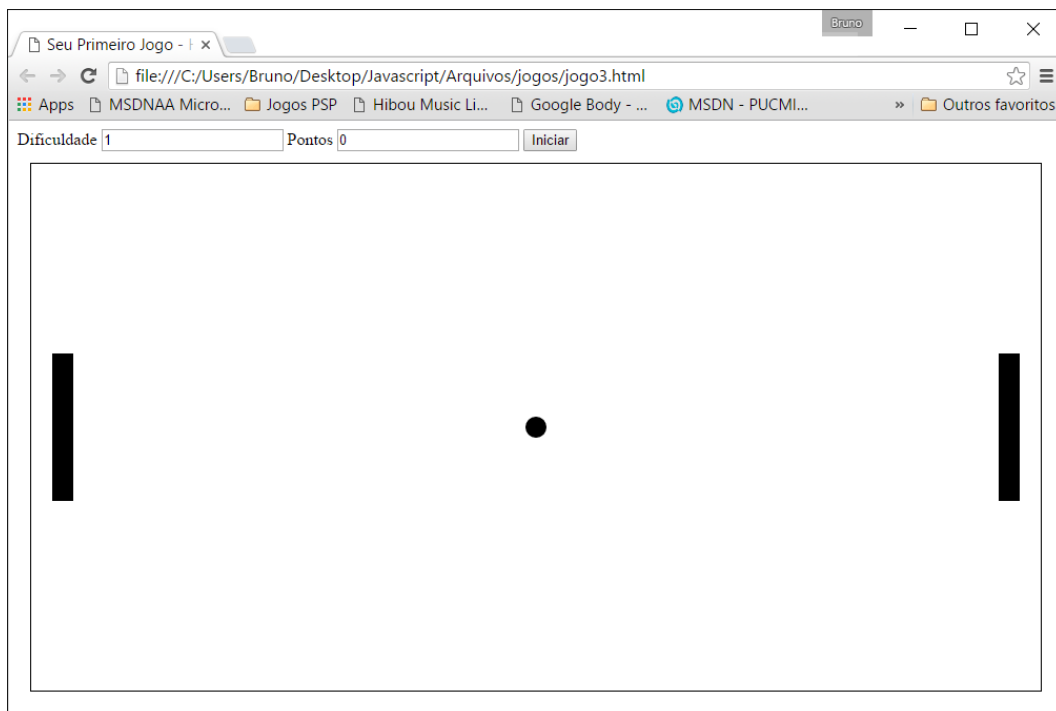
Além do “\n” podemos usar outros, confira na tabela a seguir.

Código	Descrição
\n	Quebra de linha.
\r	Retorno de carro, ENTER
\t	Tabulação TAB
\b	Backspace
\f	Form feed, adiciona um espaço

## Jogo Ping-Pong For One

Neste capítulo vamos criar um pequeno jogo para podermos praticar um pouco do que aprendemos até agora, é um jogo simples de um jogador, consiste de uma bola que inicia no centro do jogo, ao clicar no botão “iniciar” a bola

começa a se movimentar em uma direção aleatória, o objetivo do jogar é simplesmente rebater a bola, as barras do jogador podem ser movimentadas com o teclado, teclas para cima e para baixo, a medida que a bola é rebatida a velocidade aumenta e os pontos vão sendo somados.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Seu Primeiro Jogo - HTML5</title>
    <style type="text/css">
      #bola{
        width:20px;
        height:20px;
        background-color:#000;
        border-radius:10px;
      }
      #jog1, #jog2{
        width:20px;
        height:140px;
        background-color:#000;
      }
      #campo{
        width:960px;
        height:500px;
        border:1px solid #000;
      }
    </style>

    <script>

      campoX=20, campoY=40, campoW=960, campoH=500;
      dirJy=0;
      vel=5, frames=40, dificuldade=1, pontos=0;

      function iniBola(){
        if ((Math.random()*10)<5){
          dirBolaX=-1;
        }else{
          dirBolaX=1;
        }
        if ((Math.random()*10)<5){
          dirBolaY=-1;
```

```

    }else{
        dirBolaY=1;
    }
    vbola=document.getElementById("bola");
    posBolaX=470;
    posBolaY=240;
    vbola.style.top=posBolaY+"px";
    vbola.style.left=posBolaX+"px";
}

function inijog(){
    vjog1=document.getElementById("jog1");
    posJ1x=20;
    posJ1y=180;
    vjog1.style.top=posJ1y+"px";
    vjog1.style.left=posJ1x+"px";

    vjog2=document.getElementById("jog2");
    posJ2x=920;
    posJ2y=180;
    vjog2.style.top=posJ2y+"px";
    vjog2.style.left=posJ2x+"px";
}

function teclaDw(){
    var tecla = event.keyCode;
    if(tecla==38){
        //Cima
        dirJy=-1;
    }else if(tecla==40){
        //baixo
        dirJy=+1;
    }
}

function teclaUp(){
    var tecla = event.keyCode;
    if(tecla==38){
        //Cima
        dirJy=0;
    }else if(tecla==40){
        //baixo
        dirJy=0;
    }
}

function colisao(){
    //Colisão com o jog1
    if(
        (posBolaX <= posJ1x+20)&&
        ((posBolaY+20 >= posJ1y)&&(posBolaY <= posJ1y+180))
    ){
        dirBolaX=1;
        clearInterval(jogo);
        if(frames > 8){
            frames-=2;
        }
        jogo=setInterval(enterFrame,frames);
        pontos+=dificuldade;
        dificuldade+=1;
        document.getElementById("pnPts").value=pontos;
        document.getElementById("pnDif").value=dificuldade;
    }

    //Colisão com o jog2
    if(
        (posBolaX >= posJ2x-20)&&
        ((posBolaY+20 >= posJ2y)&&(posBolaY <= posJ2y+180))
    ){
        dirBolaX=-1;
        clearInterval(jogo);
        if(frames > 8){
            frames-=2;
        }
        jogo=setInterval(enterFrame,frames);
        pontos+=dificuldade;
        dificuldade+=1;
        document.getElementById("pnPts").value=pontos;
        document.getElementById("pnDif").value=dificuldade;
    }
}

```

```

    }

}

function controlajog(){
    if(dirBolaX == -1){
        posJ1y+=dirJy*10;
        vjoga=document.getElementById("jog1");
        document.getElementById("jog1").style.backgroundColor="#F00";
        document.getElementById("jog2").style.backgroundColor="#000";
        vjoga.style.top=posJ1y+"px";
        if((posJ1y+180) >= campoH+40){
            posJ1y+=(dirJy*10)*(-1);
        }else if((posJ1y) <= 0){
            posJ1y+=(dirJy*10)*(-1);
        }
    }else{
        posJ2y+=dirJy*10;
        vjoga=document.getElementById("jog2");
        document.getElementById("jog1").style.backgroundColor="#000";
        document.getElementById("jog2").style.backgroundColor="#F00";
        vjoga.style.top=posJ2y+"px";
        if((posJ2y+180) >= campoH+40){
            posJ2y+=(dirJy*10)*(-1);
        }else if((posJ2y) <= 0){
            posJ2y+=(dirJy*10)*(-1);
        }
    }
}

function controlaBola(){
    vbola=document.getElementById("bola");
    posBolaY+=(vel*dirBolaY);
    posBolaX+=(vel*dirBolaX);
    vbola.style.top=posBolaY+"px";
    vbola.style.left=posBolaX+"px";
    if(posBolaY >= 480){
        dirBolaY=-1;
    }else if(posBolaY <= 0){
        dirBolaY=1;
    }
    if((posBolaX >= 940)|| (posBolaX <= 0)){
        reset();
    }
}

function reset(){
    posBolaX=470;
    posBolaY=240;
    clearInterval(jogo);
    document.getElementById("pnIniciar").removeAttribute("disabled");
    if((Math.random()*10)<5){
        dirBolaX=-1;
    }else{
        dirBolaX=1;
    }
    if((Math.random()*10)<5){
        dirBolaY=-1;
    }else{
        dirBolaY=1;
    }
    vbola=document.getElementById("bola");
    vbola.style.top=posBolaY+"px";
    vbola.style.left=posBolaX+"px";

    vjoga=document.getElementById("jog1");
    posJ1x=20;
    posJ1y=180;
    vjoga.style.top=posJ1y+"px";
    vjoga.style.left=posJ1x+"px";

    vjog2=document.getElementById("jog2");
    posJ2x=920;
    posJ2y=180;
    vjog2.style.top=posJ2y+"px";
    vjog2.style.left=posJ2x+"px";
}

```



```

        function enterFrame(){
            controlaBola();
            controlajog();
            colisao();
        }

        function preparaJogo(){
            document.addEventListener("keydown", teclaDw);
            document.addEventListener("keyup", teclaUp);
            iniBola();
            iniJog();
        }

        function iniciaJogo(){
            frames=50;
            pontos=0;
            dificuldade=1;
            document.getElementById("pnPts").value=pontos;
            document.getElementById("pnDif").value=dificuldade;
            jogo=setInterval(enterFrame, frames);
            document.getElementById("pnIniciar").disabled="disabled";
        }
    </script>
</head>

<body onload="preparaJogo()">
    <div id="painel">
        <label>Dificuldade</label>
        <input type="text" id="pnDif" value="1">
        <label>Pontos</label>
        <input type="text" id="pnPts" value="0">
        <input type="button" value="Iniciar" id="pnIniciar" onclick="iniciaJogo()">
    </div>
    <div id="campo" style="position:absolute;top:40px;left:20px">
        <div id="bola" style="position:absolute;top:240px;left:470px"></div>
        <div id="jog1" style="position:absolute;top:180px;left:20px"></div>
        <div id="jog2" style="position:absolute;top:180px;left:920px"></div>
    </div>
</body>
</html>

```

## Objeto window

O objeto window representa a janela do browser, todos os objetos globais, variáveis, funções são automaticamente membros do objeto window.

Podemos trabalhar com o objeto window de algumas maneiras interessantes, podemos abrir ou fechar uma nova janela, manipular a posição ou o tamanho, vamos ver como criamos e manipulamos uma nova janela.

O script a seguir cria uma janela associada à variável “janela”.

```

<!doctype html>
<html lang="pt-br">
    <head>
        <title>Curso de Javascript</title>
        <meta charset="UTF-8">
        <script>

            var janela=window.open();

        </script>
    </head>
    <body>

    </body>
</html>

```

O script acima é a forma mais simples de abrir uma nova janela, porém, podemos usar alguns métodos configurar melhor a nova janela aberta.

Confira a seguir a sintaxe com os parâmetros que podem ser usados.

`window.open(url, target, especificações, replace);`

**url** = Neste parâmetro podemos informar uma url de um site ou um nome de arquivo (html, imagem, etc) que será aberto na nova janela.

**target** = Especifica como a nova janela será aberta, se será aberta em uma nova janela, ou na janela atual, veja a tabela a seguir para os valores possíveis para este parâmetro.

Parâmetro	Descrição
<code>_blank</code>	Abre em uma nova janela, separada da atual
<code>_self</code>	Abre na mesma janela, sobrepondo a página atual
<code>nome</code>	Abre a nova janela dentro de outra especificada pelo nome

**especificações** = Uma série de propriedades para configuração da janela, confira a seguir uma tabela com as propriedades possíveis.

Propriedade	Descrição
<code>channelmode=yes no 1 0</code>	Especifica se a janela vai ser exibida em modo "teatro" ou normal
<code>fullscreen=yes no 1 0</code>	Abre na mesma janela em modo tela cheia, ocupando toda tela
<code>width=pixels</code>	Especifica a largura da janela
<code>height=pixels</code>	Especifica a altura da janela
<code>left=pixels</code>	Especifica a posição horizontal da janela
<code>top=pixels</code>	Especifica a posição vertical da janela
<code>menubar=yes no 1 0</code>	Mostra ou esconde a barra de menu
<code>status=yes no 1 0</code>	Mostra ou esconde a barra de status
<code>titlebar=yes no 1 0</code>	Mostra ou esconde a barra de título
<code>toolbar=yes no 1 0</code>	Mostra ou esconde a barra de ferramentas

**replace** = Especifica se a URL cria uma nova entrada ou substitui a entrada atual no histórico, os valores para este parâmetro são true ou false.

Veja o código a seguir que abre uma nova janela e carrega o site do google dentro dela

```
var janela=window.open("http://www.google.com.br", "_blank", "width=800,height=200,top=0,left=0,menubar=0,status=0");
```

Para fechar uma janela podemos usar o método `close()`, confira a seguir o código para fechar a janela que abrimos anteriormente.

```
janela.close();
```

Para mover a janela podemos usar o método `moveTo(left,top)`, veja o exemplo a seguir que move a janela na horizontal para posição 500 e na vertical para a posição 50.

```
janela.moveTo(500,50);
```

Também podemos mudar o tamanho da janela usando o método `resizeTo()`, veja o código de exemplo que altera o tamanho da janela para 500 pixels de largura por 100 pixels de altura.

```
janela.resizeTo(500,100);
```

Como último código de exemplo para window, veja a seguir um exemplo de código com dois botões, um que abre uma nova janela e outro para fechar a janela.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

      function abreJanela() {
        janela=window.open("", "minha Janela", "width=300,height=200,top=300,left=0");
      }

      function fechaJanela() {
        janela.close();
      }

    </script>
  </head>
  <body>

    <input type="button" value="Abrir" onclick="abreJanela()">
    <input type="button" value="Fechar" onclick="fechaJanela()">

  </body>
</html>
```

## Objeto navigation

O objeto navigator contém uma série de informações importantes sobre o browser como nome código, nome, versão, etc.

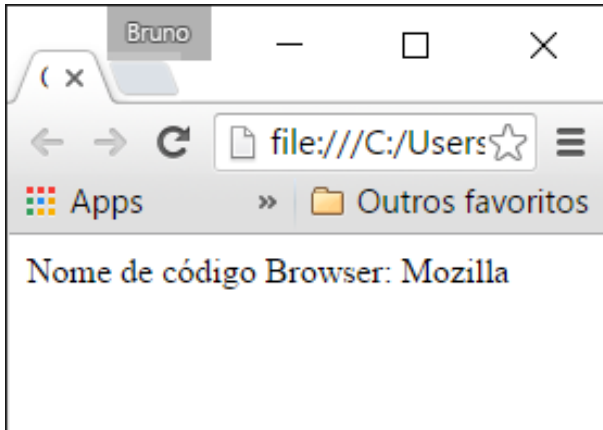
O uso do objeto é muito simples, basta usar acessar uma das propriedades disponíveis, veja a seguir um exemplo.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      document.writeln("Nome de código Browser: "+navigator.appCodeName);
    </script>
  </head>
  <body>

  </body>
</html>
```

O curioso é que todos os browseres modernos exibem o nome de código mozilla.

Veja a seguir o resultado no google Chrome.



Existem outros parâmetros que podem ser usados, veja a tabela a seguir.

Propriedade	Descrição
appName	Nome do browser, todos os browseres modernos irão retornar netscape
appVersion	Versão do browser
cookieEnabled	Se os cookies estão habilitados ou não
geolocation	Este é um objeto com informações sobre a localização geográfica
language	Idioma do browser
onLine	Se está online ou não (true / false)
platform	Qual S.O. o browser está rodando
product	Engine do browser
userAgent	Qual user agente foi enviado ao seu browser pelo servidor.

## Objeto screen

O objeto screen contém uma série de informações sobre a tela do usuário como largura, altura, cores, etc.

Veja a seguir um exemplo de uso onde iremos imprimir a largura e altura da tela.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      document.write("Tamango da tela:<br>");
      document.write("Largura: " + screen.width + "px<br>");
      document.write("Altura: " + screen.height + "px");
    </script>
  </head>
  <body>

  </body>
</html>
```

Confira a seguir a tabela com as propriedades disponíveis.

Propriedade	Descrição
availHeight	Altura do browser excluindo a barra de tarefas da janela
availWidth	Largura do browser excluindo a barra de tarefas da janela
colorDepth	Retorna a profundidade da paleta de cores
height	Altura total da janela
pixelDepth	Retorna a resolução da paleta de cores
width	Largura total da janela

## Objeto history

[http://www.w3schools.com/jsref/obj\\_history.asp](http://www.w3schools.com/jsref/obj_history.asp)

O objeto history contém informações sobre a navegação, página visitadas, vamos ver as possibilidades de uso. O código a seguir mostra quantas página estão no histórico do browser.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      document.write("Tamango do histórico: " + history.length);
    </script>
  </head>
  <body>

  </body>
</html>
```

Existem ainda três métodos interessantes para navegar pelo histórico do browser, back, forward e go, vamos ver.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      function voltar() {
        window.history.back();
      }
      function avancar() {
        window.history.forward();
      }
      function navegar() {
        num=document.getElementById("txtNum").value;
        window.history.go(num);
      }
    </script>
  </head>
  <body>
    <input type="button" onclick="voltar()" value="Voltar">
    <input type="button" onclick="avancar()" value="Avançar"><br>

    <input type="text" id="txtNum" value="0">
    <input type="button" onclick="navegar()" value="Avançar">
  </body>
</html>
```

back() = Retorna para a página anterior

forward() = Avança para próxima página

go() = Navega para a página correspondente ao valor indicado como parâmetro do método, valores negativos retornam e valores positivos avançam.

## Objeto location

O objeto location contém informações sobre a URL atual do browser.

Confira as tabelas a seguir com as propriedades e os métodos do objeto.

Propriedade	Descrição
hash	Retorna ou configura parte da url após o caractere #, inclusive o caractere #
host	Retorna ou configura o nome do host da página e o número da porta usada
hostname	Retorna ou configura o nome do host da página
href	Retorna ou configura a url da página
origin	Retorna o protocolo de origem, host name e número da porta
pathname	Retorna ou configura a url da página, caminho, após o host
Port	Retorna ou configura o número da porta da url
protocol	Retorna ou configura o protocolo da página
search	Retorna ou configura as informações "querystring", após o ? da url

Confira os métodos

Método	Descrição
assign()	Carrega um novo documento
reload()	Re-carrega o documento atual
request()	Substitui o documento atual por outro

Veja os exemplos de utilização.

```
location.assign("http://www.cfbcursos.com");
location.reload();
location.replace("http://www.cfbcursos.com.br");
```

## Adicionando, removendo e modificando elementos na página

Javascript nos permite adicionar, remover e modificar elementos HTML em tempo de execução da página, já sabemos encontrar estes elementos usando métodos como getElementById por exemplo, neste capítulo vamos dar um passo a mais, iremos aprender novos métodos importantes que irão expandir um pouco mais nosso leque de possibilidades com javascript.

Vamos aprender os seguintes métodos createElement, createTextNode, createAttribute, appendChild, insertBefore, replaceChild, remove e getAttribute.

E também as seguintes propriedades childNodes, parentNode.

### createElement - método

Vamos iniciar com um código usando o método createElement para adicionar um novo item a uma lista <ul>, vamos criar uma pequena estrutura com uma caixa de texto e um botão, quando clicar no botão o texto informado na caixa de texto será adicionado na lista.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

      function addLI(veiculo){
        if(veiculo == ""){
          alert("Informe um veículo");
          document.getElementById("txtVeiculo").focus();
        }else{
          var lista=document.getElementById("transp");
```

```

        var novoItem=document.createElement("li");
        novoItem.textContent=veiculo;
        lista.appendChild(novoItem);
    }
}

</script>
</head>
<body>

    <input type="text" value="" id="txtVeiculo">
    <button onclick="addLI(document.getElementById('txtVeiculo').value)">Adicionar</button>

    <ul id="transp">
        <li>Avião</li>
        <li>Carro</li>
        <li>Moto</li>
    </ul>

</body>
</html>

```

Vamos ver em detalhe a rotina para adicionar o novo <li>.

Primeiramente obtivemos a lista e armazenamos na variável “lista”, para facilitar o trabalho.

```
var lista=document.getElementById("transp");
```

Depois criamos um novo elemento do tipo <li> pois é o tipo de elemento que precisamos adicionar.

```
var novoItem=document.createElement("li");
```

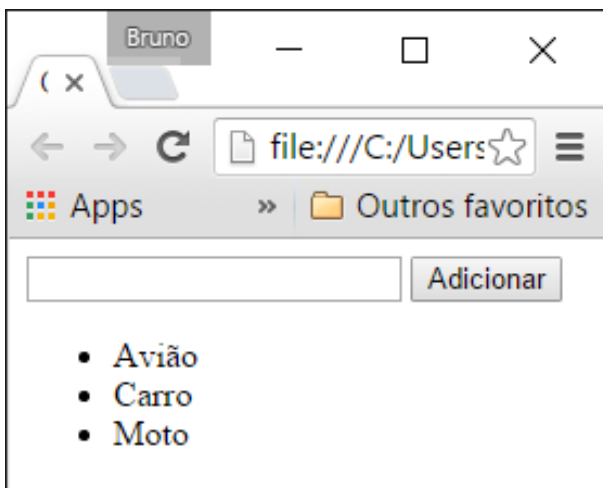
O próximo passo é definir o texto referente ao novo elemento, em nosso caso, estamos adicionando o texto passado como parâmetro na chamada da função.

```
novoItem.textContent=veiculo;
```

O último passo é adicionar o novo item <li> em nossa lista.

```
lista.appendChild(novoItem);
```

Este é o resultado do nosso código.



Veja o mesmo código onde adicionamos uma tag <p> ao <body>.

```

<!doctype html>
<html lang="pt-br">
    <head>
        <title>Curso de Javascript</title>
        <meta charset="UTF-8">
        <script>

            function addP(){
                var novoItem=document.createElement("p");

```

```

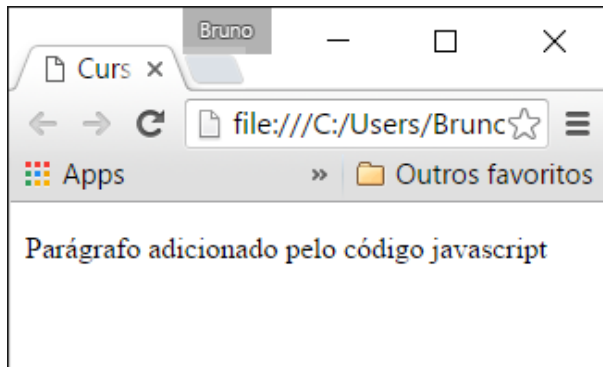
        novoItem.textContent="Parágrafo adicionado pelo código javascript";
        document.body.appendChild(novoItem);
    }

</script>
</head>
<body onload="addP()">

</body>
</html>

```

Resultado.



### *createTextNode - método*

Cria um texto que pode ser adicionado a um elemento html, veja o código onde criamos uma <div> e adicionamos um texto dentro desta div.

```

<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

        function addP(){
            var novaDiv=document.createElement("div");
            var novoTexto=document.createTextNode("Texto dentro da div");
            novaDiv.appendChild(novoTexto);
            document.body.appendChild(novaDiv);
        }

    </script>
  </head>
  <body onload="addP()">

  </body>
</html>

```

Podemos usar com qualquer elemento que use textos, veja a alteração da função que adiciona o texto ao elemento <h1>.

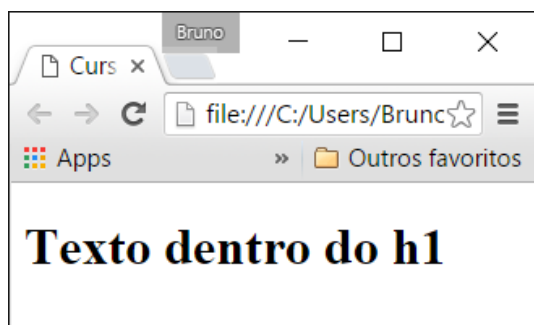
```

function addTxt(){
    var novoH1=document.createElement("h1");
    var novoTexto=document.createTextNode("Texto dentro do h1");
    novoH1.appendChild(novoTexto);
    document.body.appendChild(novoH1);
}

```

Resultado.





## *createAttribute - método*

Cria um novo atributo para adicionar ao element html, em nosso código de exemplo temos 3 tags <h1> criamos três atributos style, modificamos o valor destes atributos e configuramos nas tags h1.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

      function addP(){
        var tagsH=document.getElementsByTagName("h1");
        var at0=document.createAttribute("style");
        var at1=document.createAttribute("style");
        var at2=document.createAttribute("style");

        at0.value="color:#F00";
        tagsH[0].setAttributeNode(at0);

        at1.value="color:#0F0";
        tagsH[1].setAttributeNode(at1);

        at2.value="color:#00F";
        tagsH[2].setAttributeNode(at2);

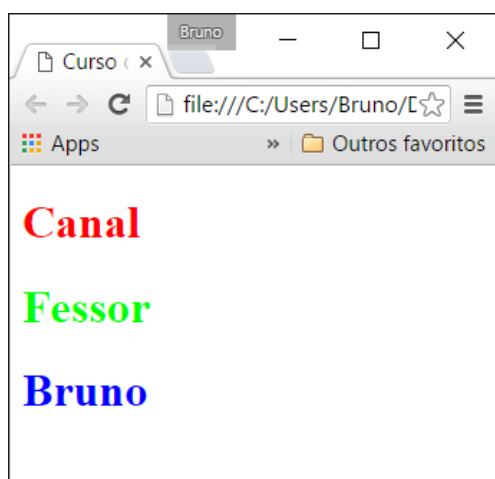
      }

    </script>
  </head>
  <body onload="addP()">

    <h1>Canal</h1>
    <h1>Fessor</h1>
    <h1>Bruno</h1>

  </body>
</html>
```

Veja o resultado do código.



Anteriormente nós já conhecemos os métodos `setAttribute`, `removeAttribute` e `hasAttribute`.

### *insertBefore - método*

Inserir um novo elemento antes de um outro elemento indicado. No código de exemplo inserimos um novo elemento `<h1>` antes do `<h1 id="t3">`, ainda adicionamos o atributo `id="t2"` neste novo `<h1>`.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

      function addP(){
        var novoH=document.createElement("h1");
        var elemento=document.getElementById("t3");
        var txtH=document.createTextNode("Fessor");
        var att=document.createAttribute("id");
        att.value="t2";
        novoH.setAttributeNode(att);

        novoH.appendChild(txtH);

        var atl=document.createAttribute("style");
        atl.value="color:#F00";
        novoH.setAttributeNode(atl);

        document.body.insertBefore(novoH,elemento);

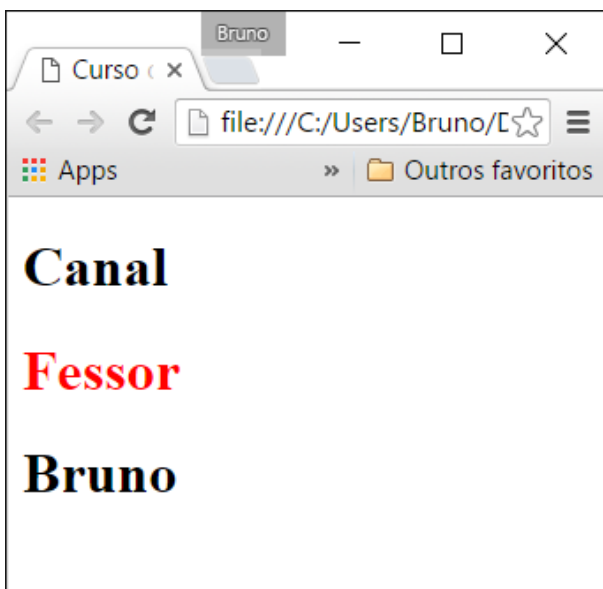
      }

    </script>
  </head>
  <body onload="addP()">

    <h1 id="t1">Canal</h1>
    <h1 id="t3">Bruno</h1>

  </body>
</html>
```

O resultado será o mostrado a seguir.



Vamos modificar o código anterior, usando a formatação das cores fora do javascript.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
```

```

<meta charset="UTF-8">
<style>
    #t1{color:#F00}
    #t2{color:#0F0}
    #t3{color:#00F}
</style>
<script>

    function addP(){
        var novoH=document.createElement("h1");
        var elemento=document.getElementById("t3");
        var txtH=document.createTextNode("Fessor");
        var att=document.createAttribute("id");
        att.value="t2";
        novoH.setAttributeNode(att);

        novoH.appendChild(txtH);

        document.body.insertBefore(novoH,elemento);

    }

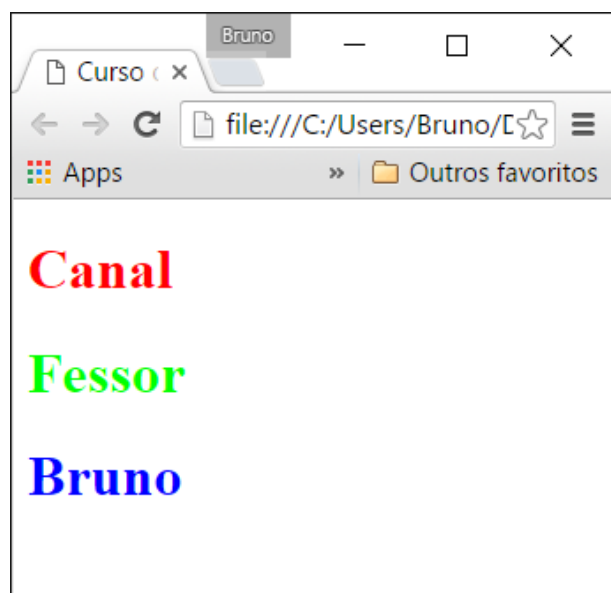
</script>
</head>
<body onload="addP()">

    <h1 id="t1">Canal</h1>
    <h1 id="t3">Bruno</h1>

</body>
</html>

```

Confira o novo resultado.



### *replaceChild - método*

Substitui um elemento por outro, simples assim, veja o código de exemplo onde substituímos uma tag <h2> por uma nova tag <h1>.

```

<!doctype html>
<html lang="pt-br">
    <head>
        <title>Curso de Javascript</title>
        <meta charset="UTF-8">
        <script>

            function addP(){
                var novoH1=document.createElement("h1");
                var txtNovo=document.createTextNode("Canal Fessor Bruno");
                novoH1.appendChild(txtNovo);
            }

```

```

        var antigoH2=document.getElementById("t1");

        document.body.replaceChild(novoH1,antigoH2);

    }

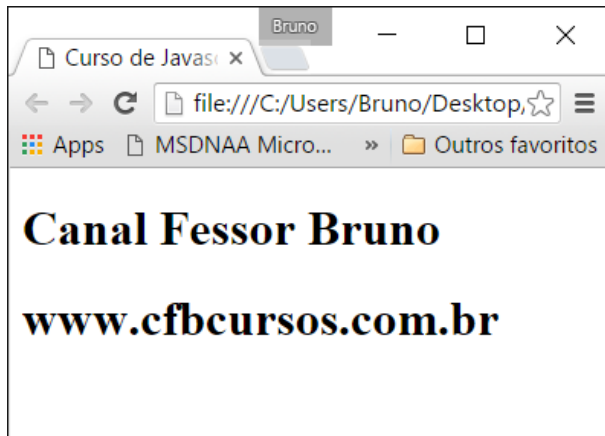
</script>
</head>
<body onload="addP()">

    <h2 id="t1">Curso de Javascript</h1>
    <h1 id="t2">www.cfbcursos.com.br</h1>

</body>
</html>

```

O resultado final.



### *childNodes – propriedade*

Esta propriedade contém todos os elementos filhos de um determinado elemento informado. Em nosso código de exemplo obtivemos todos os elementos existentes no <body>.

```

<!doctype html>
<html lang="pt-br">
    <head>
        <title>Curso de Javascript</title>
        <meta charset="UTF-8">
        <script>

            function elementosDoBody(){
                var elementos=document.body.childNodes;
                var nome = "";
                var i;
                for(i = 0; i < elementos.length; i++){
                    nome=nome + elementos[i].nodeName + "<br>";
                }
                document.getElementById("txtElementos").innerHTML = nome;
            }

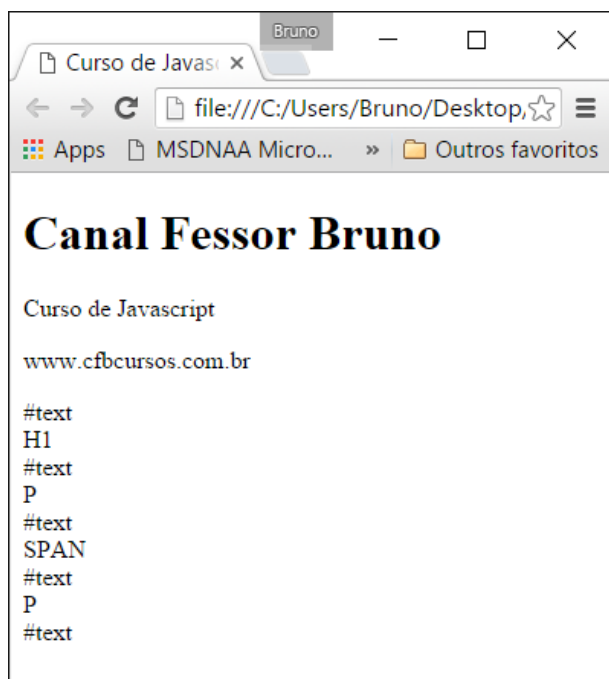
        </script>
    </head>
    <body onload="elementosDoBody()">

        <h1>Canal Fessor Bruno</h1>
        <p>Curso de Javascript</p>
        <span>www.cfbcursos.com.br</span>
        <p id="txtElementos"></p>

    </body>
</html>

```

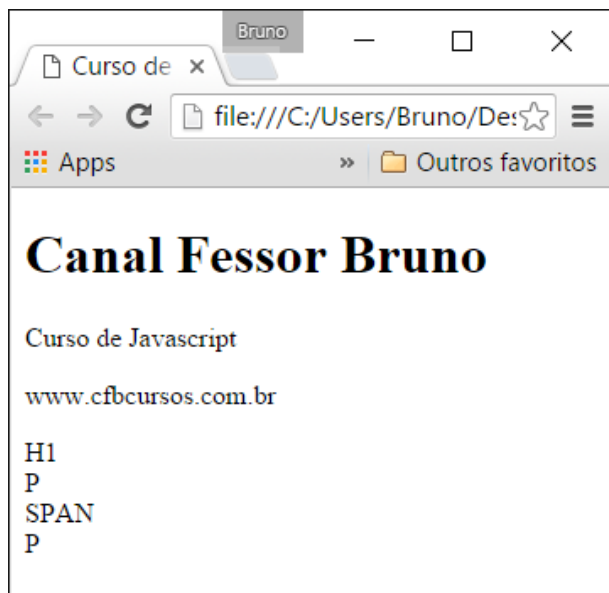
O resultado.



Podemos alterar o loop for para imprimir somente as tags e não os textos, então, altere o FOR conforme o código a seguir.

```
for(i = 1; i < elementos.length; i+=2){
    nome=nome + elementos[i].nodeName + "<br>";
}
```

E confira o novo resultado.



### *parentNode – propriedade*

Esta propriedade contém o elemento pai do elemento informado, em nosso código de exemplo buscamos o elemento pai que é o <ul> do elemento <li> informado.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>

      function primeiroElemento() {
        var elemento=document.getElementById("item1").parentNode.nodeName;
        alert(elemento);
      }

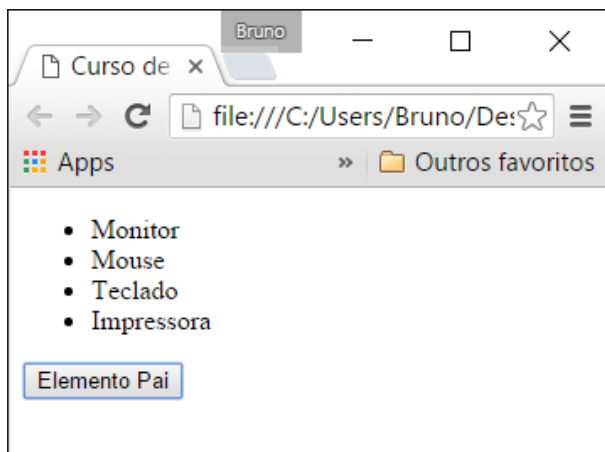
    </script>
  </head>
  <body>

    <ul id="info">
      <li id="item1">Monitor</li>
      <li id="item2">Mouse</li>
      <li id="item3">Teclado</li>
      <li id="item4">Impressora</li>
    </ul>

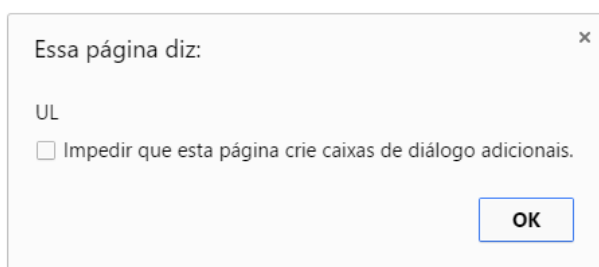
    <button onclick="primeiroElemento()">Elemento Pai</button>

  </body>
</html>
```

Resultado.



Ao clicar no botão é mostrada a caixa de mensagem com o elemento pai.



## *remove() – método*

O método `remove()` literalmente remove um elemento em nossa página, tem uma implementação muito simples, em nosso primeiro código de exemplo vamos implementar um script para remover todas as tags `<p>` do documento, da última até a primeira.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
```

```
<script>
    function removerP() {
        var ps=document.getElementsByTagName("p");
        var qtde=ps.length;
        if(qtde > 0){
            ps[qtde-1].remove();
        }else{
            alert("Não existem mais elementos P");
        }
    }
</script>
</head>
<body>

    <button onclick="removerP()">Remover P</button>

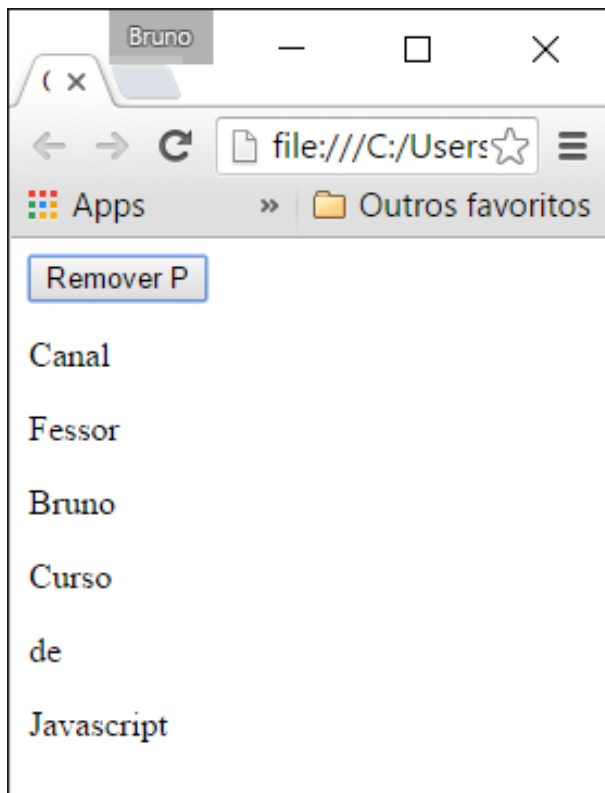
    <p>Canal</p>
    <p>Fessor</p>
    <p>Bruno</p>
    <p>Curso</p>
    <p>de</p>
    <p>Javascript</p>

</body>
</html>
```

No código acima obtivemos todas as tags `<p>` usando o método “`getElementsByTagName`”, pegamos a quantidade destes elementos com o parâmetro “`length`” e se o tamanho for maior que zero, usamos o método `remove` para remover o último `<p>`.

Veja bem, na variável “`qtde`” temos a quantidade de elementos `<p>`, que inicialmente é 6, porém o vetor inicia com índice zero, então no vetor temos de 0 a 5, por isso usamos `[qtde-1]`.

O resultado desta página pode ser conferido a seguir.



Sempre que clicar no botão “Remover P” a última tag `<p>` será removida, até que não sobre mais nenhuma.

Uma outra forma de utilização do método `remove` é indicar qual elemento desejamos remover como parâmetro do método, veja o código a seguir onde removemos os últimos elementos de uma lista até que não sobre mais nenhum.

```

<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      function removerItem(){
        var lista=document.getElementById("info");
        var tam=lista.length;
        if(tam > 0){
          lista.remove(tam-1);
        }else{
          alert("Não existem mais itens na lista");
        }
      }
    </script>
  </head>
  <body>

    <button onclick="removerItem()">Remover item</button>

    <select id="info">
      <option>Monitor</option>
      <option>Teclado</option>
      <option>Mouse</option>
      <option>Impressora</option>
      <option>Microfone</option>
      <option>Caixa de som</option>
    </select>

  </body>
</html>

```

Confira agora uma variação do programa acima, onde removemos o item que foi selecionado.

```

<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      function removerItem(){
        var lista=document.getElementById("info");
        var tam=lista.length;
        if(tam > 0){
          lista.remove(lista.selectedIndex);
        }else{
          alert("Não existem mais itens na lista");
        }
      }
    </script>
  </head>
  <body>

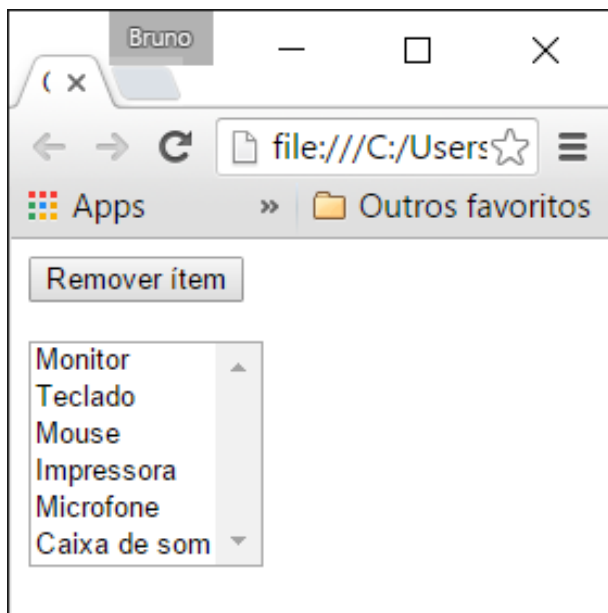
    <button onclick="removerItem()">Remover item</button><br><br>

    <select id="info" size="6">
      <option>Monitor</option>
      <option>Teclado</option>
      <option>Mouse</option>
      <option>Impressora</option>
      <option>Microfone</option>
      <option>Caixa de som</option>
    </select>

  </body>
</html>

```





### *getAttribute() – método*

O método `getAttribute` retorna o valor de um atributo especificado, em nosso script de exemplo vamos mostra duas divs formatadas como quadrados vermelho e azul, a formatação da cor é feita propositalmente com classe, ao clicar na div é mostrada um alert informando o valor do atributo `class` da div, veja o código.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <style>
      div{
        position:absolute;
        top:0px;
        width:100px;
        height:100px;
      }

      .dv1{
        background-color:#F00;
        left:50px;
      }

      .dv2{
        background-color:#00F;
        left:200px;
      }
    </style>
    <script>
      function mostraAtributo(id){
        var dv=document.getElementById(id);
        var valAtt=dv.getAttribute("class");
        alert("Usa a classe " + valAtt);
      }
    </script>
  </head>
  <body>

    <div id="d1" class="dv1" onclick="mostraAtributo(this.id)"></div>
    <div id="d2" class="dv2" onclick="mostraAtributo(this.id)"></div>

  </body>
</html>
```

Resultado.



## activeElement - propriedade

A propriedade `activeElement` contém o elemento que está selecionado, que possui o foco no momento, observe o código de exemplo, temos três elementos em nossa página e uma função que usa a propriedade `activeElement` retornando o objeto selecionado para a variável "obj", clique nos elementos para testar.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      function ativo() {
        var obj = document.activeElement.tagName;
        document.getElementById("txtAtivo").innerHTML = obj;
      }
    </script>
  </head>
  <body onclick="ativo()">
    <p>Body</p>
    <input type="text"><br><br>
    <button>Botão</button>

    <p id="txtAtivo"></p>
  </body>
</html>
```

## hasFocus - método

O método `hasFocus` retorna `true` se o documento possui o foco, também possibilita saber se o usuário está trabalhando com múltiplas janelas ou abas, até mesmo se alternou para outro aplicativo saindo do navegador deixando-o em segundo plano.

Nosso programa de exemplo verifica todo momento se o documento está ativo, em foco, enviando a mensagem se está ou não.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      function ativo() {
        var obj = document.getElementById("txtAtivo");
        if (document.hasFocus()) {
          obj.innerHTML = "O documento está em foco.";
        } else {

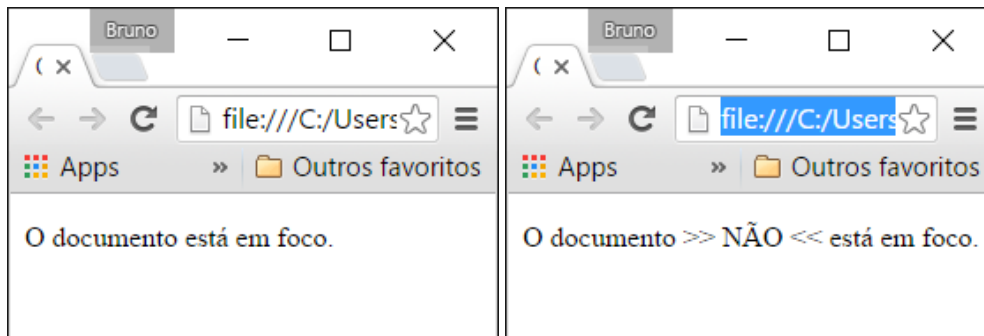
```

```

        obj.innerHTML = "O documento >> NÃO << está em foco.";
    }
    }
    var intervalo=setInterval("ativo()", 100);
</script>
</head>
<body onclick="ativo()">
    <p id="txtAtivo"></p>
</body>
</html>

```

Para testar nosso script, restaure o browser, clique fora e dentro da janela para ver as mensagens.



## Datasets

Dataset é um recurso muito interessante, permite que adicionemos dados a elementos HTML, como se fossem constantes, por exemplo podemos criamos um elemento e definimos uma série de atributos que funcionarão como informações deste elemento, que pode ser facilmente obtido por javascript.

Veja um exemplo simples, onde adicionamos três dados vel, pot e cor, note que basta adicionar “data-” antes do nome do atributo.

```
<p id="pCar1" data-vel="320" data-pot="400" data-cor="Vermelho">Carro 1</p>
```

Para pegar os valores usamos a seguinte sintaxe.

```

var varVel=document.getElementById("pCar1").dataset.vel;
var varPot=document.getElementById("pCar1").dataset.pot;
var varCor=document.getElementById("pCar1").dataset.cor;

```

Em nosso programa de exemplo criamos três botões com três atributos cada e uma função para pegar o valor destes atributos e mostrar em um alert.

```

<!doctype html>
<html lang="pt-br">
<head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
        function mostraVal(idObj){
            var ve=document.getElementById(idObj).dataset.vel;
            var pt=document.getElementById(idObj).dataset.pot;
            var cr=document.getElementById(idObj).dataset.cor;
            alert("Velocidade Máxima: " + ve + "\nPotência: " + pt + "\nCor: " + cr);
        }
    </script>
</head>
<body>
    <button id="btCar1" data-vel="320" data-pot="400" data-cor="Vermelho" onclick="mostraVal(this.id)">Carro 1</button>
    <button id="btCar2" data-vel="180" data-pot="150" data-cor="Amarelo"onclick="mostraVal(this.id)">Carro 2</button>
    <button id="btCar3" data-vel="120" data-pot="80" data-cor="Azul" onclick="mostraVal(this.id)">Carro 3</button>
</body>
</html>

```

## scrollIntoView

Este método rola a página até que o elemento indicado como parâmetro esteja no topo da janela, veja o código de exemplo.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <style>
      h2{margin-bottom:750px;}
    </style>
    <script>
      function rola(idObj){
        var obj=document.getElementById(idObj).scrollIntoView();
      }
    </script>
  </head>
  <body>
    <button onclick="rola('idTopo')">Topo</button>
    <button onclick="rola('idMeio')">Meio</button>
    <button onclick="rola('idFim')">Fim</button>

    <h2 id="idTopo">topo</h2>

    <button onclick="rola('idTopo')">Topo</button>
    <h2 id="idMeio">meio</h2>

    <button onclick="rola('idTopo')">Topo</button>
    <h2 id="idFim">fim</h2>
  </body>
</html>
```

Configuramos via CSS para que as margens inferiores dos elementos <h2> fiquem grande para que possamos rolar a janela sem precisar de um texto grande.

Dentro da função usamos o método scrollIntoView no elemento indicado pelo id passado como parâmetro "idObj".

Por último associamos a função aos botões.

**OBS:** O padrão é que o elemento role até que fique visível no topo da tela, mas, podemos indicar que o elemento role até que fique visível na parte de baixo da tela, basta passar o parâmetro false para o método, veja o exemplo a seguir.

```
var obj=document.getElementById(idObj).scrollIntoView(false);
```

## Atributo hidden

É muito comum em uma rotina de programação em Javascript precisarmos ocultar e ou exibir elementos das páginas, essa tarefa é bem simples em javascript, basta usarmos o atributo hidden passando true ou false. No script de exemplo temos dois botões que chamam as funções de mostrar e ocultar respectivamente.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      function ocultar(idObj){
        var obj=document.getElementById(idObj).hidden=true;
      }

      function mostrar(idObj){
        var obj=document.getElementById(idObj).hidden=false;
      }
    </script>
  </head>
  <body>
    <button onclick="ocultar('idTopo')">Ocultar</button>
    <button onclick="mostrar('idTopo')">Mostrar</button>

    <h2 id="idTopo">topo</h2>
  </body>
</html>
```

```

    }
</script>
</head>
<body>
    <button onclick="mostrar('idH2')">Mostrar</button><button
onclick="ocultar('idH2')">Ocultar</button>
    <h2 id="idH2">Canal Fessor Bruno</h2>

</body>
</html>

```



## Código javascript externo – Arquivos .js

Podemos trabalhar com os scripts em um arquivo externo do tipo “.js”, isso traz mais organização ao nosso trabalho, pois, desta maneira podemos separar o código HTML do código javascript da mesma forma que podemos fazer com CSS, não é difícil trabalhar com scripts externos veja o exemplo.

### aula.html

```

<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script src="scripts.js"></script>
  </head>
  <body>

    <button onclick="mostraAlert()">Mostrar alert</button>

  </body>
</html>

```

### scripts.js

```

function mostraAlert() {
    alert("Arquivo de script externo");
}

```

No arquivo scripts.js iremos inserir todo conteúdo relacionado a javascript da mesma forma que utilizamos dentro do arquivo html.

## Cookies

Cookies são pequenos arquivos de texto criados no computador de quem está acessando o site para armazenamento de informações.

Javascript pode criar, ler e remover cookies de forma simples usando a propriedade cookies do elemento document.

Veja o exemplo a seguir que mostra como trabalhamos com cookies de forma simples.

```
<!doctype html>
<html lang="pt-br">
  <head>
    <title>Curso de Javascript</title>
    <meta charset="UTF-8">
    <script>
      function criarCookie(nome, valor, expira) {
        var dtExpira = "expires= Tue, 01 Jan 2115 12:00:00 UTC ";
        document.cookie = nome + "=" + valor + "; " + dtExpira;
      }

      function lerCookie(nome) {
        var vnome = nome + "=";
        var ca = document.cookie.split(';');
        for(var i=0; i<ca.length; i++) {
          var c = ca[i];
          while (c.charAt(0)==' ') c = c.substring(1);
          if (c.indexOf(vnome) == 0) return c.substring(vnome.length,c.length);
        }
        return "";
      }

      function iniCookie() {
        var username=lerCookie("username");
        if (username!="") {
          alert("Bem vindo novamente " + username);
        }else{
          username = prompt("Digite seu nome:", "");
          if (username != "" && username != null) {
            criarCookie("username", username, 365);
          }
        }
      }
    </script>
  </head>
  <body onload="iniCookie()">

  </body>
</html>
```

OBS: Para deletar um cookie, basta configurar a data de expiração para uma data passada, como no exemplo a seguir.

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC";
```

Com nosso código ao abrir a página se o cookie não existir será criado com data de expiração para terça-feira primeiro de janeiro de 2115 as 12:00 horas.

Quando você abrir a página novamente o cookie já existirá e será mostrada a mensagem com seu nome “Bem vindo novamente nome digitado”.

Por motivos de segurança a maioria dos broseres não permite a criação de cookies off-line.

## Considerações finais

Neste material você aprendeu sobre o universo Javascript, existem inúmeras possibilidade de utilização deste conteúdo, a criatividade do programador é fundamental, claro que você precisa de todo conhecimento técnico, mas só a técnica não é o bastante para criar recursos interessantes, conseguir juntar todo esse trabalho para criar uma funcionalidade, este é o desafio, mas como chegar a este nível? A resposta é simples, praticando, quanto mais prática você tiver mais você vai ter capacidade de misturar o conteúdo para criar boas aplicações.

Portanto é hora de se animar e criar seus próprios scripts para suas páginas.