



Apostila JavaScript

SUMÁRIO

Introdução.....	3
Sintaxe	3
Caixas de Diálogo ou de Mensagem	3
Alert.....	3
Confirm	4
Prompt	5
Tipos de dados, Variáveis, atribuições e Operadores	6
Tipos de variáveis	6
Criando variáveis	8
Tratando os dados	9
NUMBER	9
S.LENGTH	10
String.ToupperCase() E String.ToLowerCase()	11
Operadores.....	12
Aritméticos	12
Atribuição	13
Atribuição simples.....	13
Autoatribuição	13
Incremento	14
Relacionais	14
Identidade.....	15
Lógicos.....	16
Blocos Condicionais	18
Funções.....	19
INVOCANDO.....	20
Eventos.....	21
Bloco de repetições	23
FOR	23
WHILE (Enquanto)	24
Objeto Math	25
Data e Hora	25
Bônus:	26
Array	26
Acessando elementos do array	26

Introdução

JavaScript é uma linguagem de programação interpretada, baseada em orientação a objetos utilizada para *client-side* (utilizado mais do lado do cliente, ou seja, você!) principalmente em navegadores web, mas também em outros ambientes como *Node.js* ou *Apache CouchDB*. É uma linguagem de script multiplataforma.

Ele não é mantido pelo W3C, ele é uma linguagem criada e mantida pela ECMA. Eles mantêm uma documentação da linguagem no site deles, mas a melhor documentação ainda são os materiais que você pode encontrar na web mesmo e claro por essa apostila.

Um **script** pode ser considerado um pequeno programa que é desenvolvido para executar pequenas funções em uma página web. Sua característica de orientação a objetivos provavelmente influenciou seu nome: “Javascript” seria uma espécie de linguagem Java para pequenos programas executados no browser.

Apesar deste aspecto, a linguagem possui finalidades bem distintas quando comparada ao Java, que ao contrário desta, é utilizada em “server-side”.

Algumas características do Javascript:

- Exerce controle sobre o HTML e o CSS para manipular comportamentos de páginas web.
- É independente de plataforma
- Permite a criação de funcionalidades para sites que não podem ser criadas com HTML e CSS.
- O código escrito em Javascript é embutido no próprio arquivo HTML (basta que o programador utilize a tag **<script>** no momento de escrever as linhas em Javascript).
- Utiliza uma sintaxe familiar, para quem já utiliza Java, C ou C++.
- Seu modelo de objetos é baseado em protótipos (e não em classes, como é o caso do Java).
- Suporta funções sem requisições especiais de declaração.
- É um tipo de programação dirigida por eventos: possibilita a criação de trechos de programa que respondem a eventos específicos, tais como um clique em um botão.

Sintaxe

```
<script>  
Código JavaScript  
</script>
```

O código JavaScript deve ser escrito dentro da **tag script** e pode ser adicionado em qualquer local do seu site, mas como um bom padrão de recomendação, é interessante adicionar o código JS no final da página HTML, logo acima do fechamento da **tag body** (**</body>**)

Caixas de Diálogo ou de Mensagem

Alert

O *alert* é uma das mais simples caixas de diálogo de se fazer, com uma aparência simples e intuitiva. Sua principal função é mostrar ao usuário uma mensagem e um botão de confirmação (ok) de que o usuário tenha visto a mensagem. Para chamar essa função, basta utilizarmos o código `alert()`, que receberá uma string (mensagem que será exibida ao usuário).

Reza a lenda que o programador que não fizer no seu primeiro código com as instruções que escreva na tela o simples **Hello World ou Olá Mundo**. Não conseguirá entender ou mesmo não será um bom profissional na área da programação.

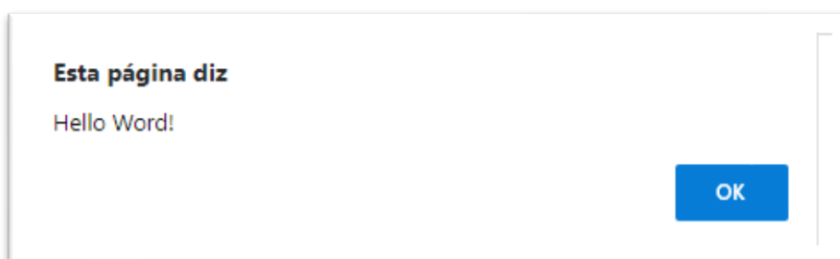


Pode ser uma Lenda urbana criada por algum programador supersticioso, mas por via das dúvidas não vamos ficar de fora dessa, ok?

Para entendermos melhor veja o exemplo do código abaixo:

```
<html>
<head>...</head>
<body>
  <script>
    alert("Hello World!");
  </script>
</body>
</html>
```

Note que o *alert* é declarado dentro do **head** com a utilização da tag **script**. O resultado será a mensagem **“Hello World”** e em seguida quando você clicar em Ok ela some.



Pronto, nos livramos da maldição e vamos sim aprender essa linguagem!

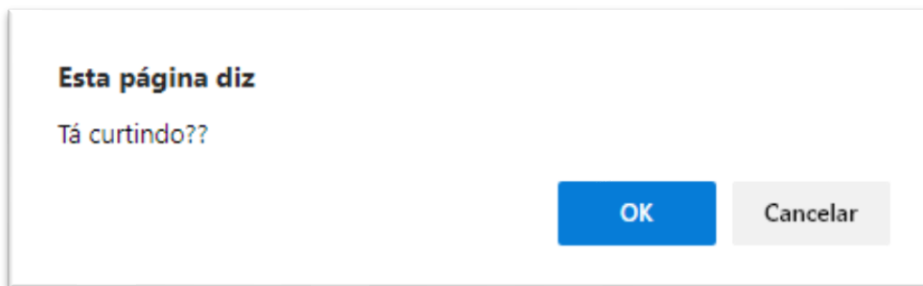
Confirm

O *confirm* cria uma mensagem com duas opções para o usuário: **Ok e Cancelar**.

Com ele é possível realizar comandos com condições e identificar o que o usuário clicou e encaminha para uma tela específica ou não. Para você entender melhor, veja o código abaixo e o seu resultado.

```
<html>
<head>...</head>
<body>
  <script>
    confirm("Você está curtindo?");
  </script>
</body>
</html>
```

Resultado:



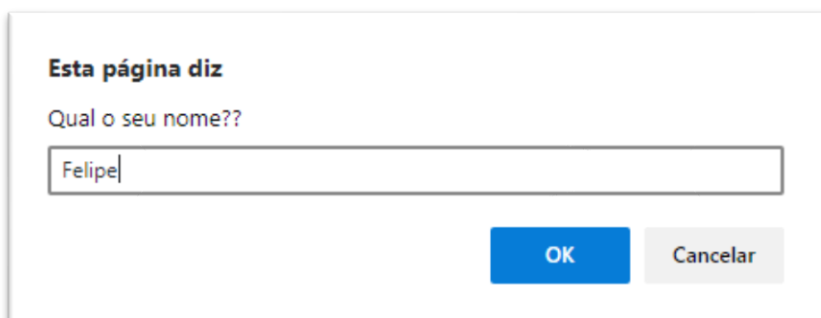
Prompt

O *prompt* apresenta uma mensagem e uma caixa de texto para que o usuário digite alguma coisa, vai depender do que você solicita. A partir desse texto digitado, o site pode apresentar alguma mensagem personalizada, realizar um cálculo com o valor digitado etc.

Veja o código e o resultado utilizando o *prompt*:

```
<html>
<head>...</head>
<body>
  <script>
    prompt("Qual o seu nome?");
  </script>
</body>
</html>
```

Resultado:



Obs.: Você pode adicionar comentários no seu código JavaScript utilizando o `//` para comentar apenas uma linha ou o `/*...*/`.

Tipos de dados, Variáveis, atribuições e Operadores

Variável é um dos conceitos mais importantes no estudo de programação, independentemente da plataforma ou linguagem utilizada. Uma variável referencia a um espaço na memória do computador utilizado para guardar informações que serão usadas em seus programas.

Para elucidar o conceito, imagine que a memória de seu computador é um armário com 100 gavetas e você guarda cada tipo de objeto em uma gaveta diferente. Provavelmente você vai querer criar etiquetas para referenciar o que guarda em cada gaveta.



Uma variável pode ter o seu valor alterado durante a execução de um programa. Ainda na analogia anterior, suponha que há uma gaveta "Meias" com 2 meias; se necessário, pode-se adicionar mais 5 meias e, caso seja um colecionador, até 200. Assim a gaveta "Meias" muda o seu valor e você terá que alterar o número nas etiquetas.

Tipos de variáveis

Quando falamos em tipos de variáveis, temos as linguagens chamadas de fortemente tipadas e fracamente tipadas.

Em linguagens **fortemente tipadas**, definimos o tipo da variável no momento de sua criação. Exemplos de linguagens do tipo: Java; C; C++. Em linguagens **fracamente tipadas**, não precisamos definir o tipo da variável, ela é tipada automaticamente quando recebe um valor. Exemplos de linguagens do tipo: Python; Ruby; **Javascript**.

- Int - Variáveis com valores inteiros.

- var idade = 17;
- var graus = -3;
- var pontos = 0;
- var numeroGrande = 2000009283;

Obs: em linguagens de programação utiliza-se "." (ponto final) para separar casas decimais em vez de "," (vírgula).

- Float - Variáveis com ponto flutuante ou casas decimais.

- var peso = 32.59345;
- var PI = 3.14;
- var meu_saldo = -1034.32

- String - Variáveis de texto, normalmente chamada de "cadeia de caracteres". Os valores desse tipo são atribuídos utilizando aspas duplas (") ou aspas simples (') como delimitador.

- var nome = "Gabriel Mendonça";
- var data_nascimento = "17 de junho de 1988";
- var email = "gabriel@host2.com.br";
- var tempo = "20s";

Obs: Tudo o que é declarado entre os delimitadores (") ou (') é entendido como parte da string, mesmo que sejam números.

- Booleanos - Tipo de dado de dois valores: "true" (verdadeiro) ou "false" (falso).

- var verdadeiro = true;
- var verdadeiro2 = 1;
- var falso1 = false;
- var falso2 = 0;
- var falso3 = null

- Arrays - Um array referência a vários espaços na memória. É um conjunto de valores e/ou variáveis organizadas por índice (que pode ser um valor inteiro ou string). O entendimento desse tipo é muito importante.

- Retomando a analogia inicial: imagine que em uma gaveta você queira armazenar bebidas. Nela teria suco, refrigerante, café, água mineral... Perceba que esses itens formam uma lista e que todos os itens são bebidos, ou seja, itens de uma mesma categoria. Com o uso de arrays evitamos declarar diversas variáveis para um mesmo grupo (nada de bebida1, bebida2!), acessando os itens pelos seus respectivos índices: bebida[1], bebida[2]. Se você perceber, declarar várias variáveis para uma mesma coisa usaria desnecessariamente várias gavetas, enquanto declarar um **array** para armazenar todas as "mesmas coisas" usaria apenas uma.

JavaScript é um tipo de linguagem de programação **case sensitive**, ou seja, faz distinção entre minúsculas e maiúsculas. Então, as variáveis *nome* e *Nome* apesar de terem as mesmas letras um é todo em minúsculo "nome" e o outro começa com a primeira letra maiúscula "Nome".

Criando variáveis

Para criar uma variável utiliza-se **var** (opcional) e, para determinar o seu valor, o operador de **atribuição** (=). Para facilitar a compreensão do código, deve-se sempre escolher um nome que identifique o tipo de dado a ser armazenado, por exemplo, se você quer armazenar o nome de alguma pessoa, é recomendado você criar a variável com a palavra "nome".

```
<html>
<head>...</head>
<body>
  <script>
    var nome="Felipe"
  </script>
</body>
</html>
```

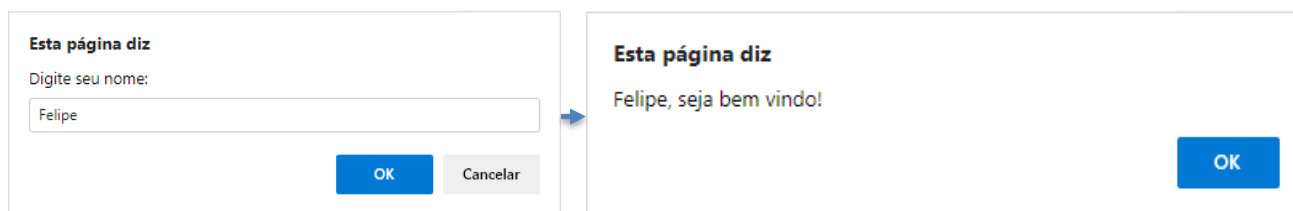
Para a atribuição de String ou Caracteres é necessário utilizar a delimitação de aspas duplas ""

Digite na ferramenta que você está utilizando o código abaixo e rode no seu navegador.

```
<html>
<head>...</head>
<body>
  <script type="text/javascript">
    var nome = prompt('Digite seu nome: ');
    alert(nome + ' , seja bem vindo!');
  </script>
</body>
</html>
```

Lembre de sempre salvar o arquivo e abrir no navegador o seu arquivo **HTML**.

Resultado:



Nesse exemplo, declaramos a variável "**nome**", onde guardamos o nome que foi solicitado ao usuário através do método **prompt()** (que eu digitei aqui o nome Felipe). Após o nome informado ser armazenado (isso é, se tornado valor da variável), uma mensagem de boas-vindas é apresentada ao usuário através do método **alert()** utilizando o nome armazenado na variável.

Existem diversas formas que você pode utilizar as variáveis com as caixas de mensagens.


```
11 <script type="text/javascript">
12   /* Este é um script para cálculo de idade! */
13
14   // Declara o ano atual para fazer o cálculo
15   var anoAtual = 2020;
16
17   // Pede que o usuário digite o ano em que nasceu
18   var anoNascimento = prompt('Digite o ano em que você nasceu.');
```

Note que adicionei o código direto do meu programa, dessa forma facilita a visualização para você como devemos organizar o código

```
19
20   // Calcula a idade do usuário e armazena na variável idade
21   var idade = anoAtual - anoNascimento;
22
23   // Mostra ao usuário a idade que ele possui
24   alert("Sua idade é: " + idade + " anos");
25 </script>
```

No código acima temos algumas coisas novas: operadores **aritméticos de soma e subtração (+, -)** na linha 21 do código, o operador de **concatenação (+, para juntar strings)** na linha 24 e de **comentários** (// para linha única e /* */ para múltiplas linhas).

Um comentário é um trecho no código que não é executado, e por isso serve como um espaço para explicações e descrições relacionadas ao código ou até mesmo para evitar a execução de um bloco de código. Veja o resultado abaixo:

É importante todo código que você visualizar aqui ou em outros materiais você praticar aí no seu computador. Então repliquem e modifiquem a sua maneira para ver como o código se comporta!!

Tratando os dados

NUMBER

A caixa de diálogo **prompt()** quando é solicitada, retorna por padrão uma **String** independente se for digitado um número ou uma palavra. Veja no exemplo abaixo de uma soma de dois números digitados, a intenção do código é pedir dois números ao usuário e apresentar a soma dos dois.

```
11 <script type="text/javascript">
12
13   var numero1 = prompt('Digite o primeiro número');
14   var numero2 = prompt('Digite o segundo número');
```

Note que adicionei o código direto do meu programa, dessa forma facilita a visualização para você como devemos organizar o código

```
15
16   var soma = numero1 + numero2;
17
18   // Mostra ao usuário a idade que ele possui
19   alert("A soma dos números é:" + soma);
20 </script>
```

The diagram illustrates a sequence of three JavaScript prompts. The first prompt, titled 'Esta página diz', asks 'Digite o primeiro número' and shows the input '20'. An arrow points to the second prompt, which asks 'Digite o segundo número' and shows the input '5'. A second arrow points to a third prompt titled 'Esta página diz' showing the result 'A soma dos números é: 205'. The '205' is circled, indicating the error where the numbers were concatenated instead of summed.

Note que o número apresenta deveria ser o 25 (20+5), porém ele mostrou o 205. Esse problema decorre por conta do tipo de dados que o **prompt()** recebe, mesmo que tenha sido digitado dois números ele entende como caracteres qualquer e no código ele simplesmente **concatena** (junta). Para corrigir nós devemos forçar o código a interpretar o que foi digitado no **prompt()** como número, veja no código corrigido abaixo:

```

11 <script type="text/javascript">
12
13     var numerol = Number(prompt('Digite o primeiro número'));
14     var numero2 = Number(prompt('Digite o segundo número'));
15
16     var soma = numerol + numero2;
17
18     // Mostra ao usuário a idade que ele possui
19     alert("A soma dos números é: " + soma);
20 </script>

```

Note que o **prompt** ficou dentro das parênteses da função **Number** destacado de vermelho no código

Para forçar o JS a entender como número, basta inserir antes do **prompt()** a função **Number()**. Para visualizar os outros tipos de conversões acesse [ESSE LINK](#).

S.LENGTH

A propriedade **length** tem como responsabilidade retornar a quantidade de caracteres de uma string ou o tamanho de um array. Caso a string ou o array esteja vazio, é retornado o valor 0. No exemplo a seguir demosntro como obter o tamanho de uma string:

```

11 <script type="text/javascript">
12     var nome = "Maria Eduarda de Almeida";
13     var tamanho = nome.length;
14     alert (O nome tem `${tamanho} letras`)
15 </script>

```

O resultado apresentado será 24 letras pois ele conta também os espaços entre as palavras. Veja o resultado abaixo:

Esta página diz

O nome tem 24 letras

OK

String.ToupperCase() E String.ToLowerCase()

O método **toLowerCase()** retorna o valor da string convertido para minúsculo. **toLowerCase()** não altera o valor da string em que o método é chamado. O **toUpperCase()** converte todo o conteúdo em maiúsculo e também não altera o valor da string. Veja no código abaixo e seu resultado:

```
11 <script type="text/javascript">
12     var nome = "Maria Eduarda de Almeida";
13     alert(nome.toUpperCase())
14     alert(nome.toLowerCase())
15     document.write(nome.toUpperCase())
16 </script>
```

Resultado:

The diagram illustrates the execution of the JavaScript code. It shows two sequential alert boxes. The first alert displays 'Esta página diz' followed by 'MARIA EDUARDA DE ALMEIDA'. The second alert displays 'Esta página diz' followed by 'maria eduarda de almeida'. Below these, a browser window shows the document content as 'MARIA EDUARDA DE ALMEIDA', which was written to the page by the `document.write()` command. Arrows indicate the flow from the first alert to the second, and then from the second alert to the browser window.

Primeiro o site vai mostrar o nome com todas as letras maiúsculas utilizando o **nome.toUpperCase()**, em seguida a mensagem com o nome com todas as letras minúsculas utilizando o **nome.toLowerCase()**. Por fim adicionamos uma coisa nova, escrever no site a variável com as letras maiúsculas.

Bônus:

O JavaScript tem uma técnica que facilita a concatenação de caracteres com variáveis, isso pode te poucar um pouco de trabalho já que se for um código com muitas variáveis pode ter um pouco de confusão para apresentar a mensagem, veja na linha de código abaixo:

```
11 <script type="text/javascript">
12
13     var nome = "Felipe"
14     var idade = 27
15     var nota = 6.5
16     var matricula = 8754
17
18     alert("O aluno " + nome + " tem " + idade + " anos de matrícula " + matricula + " teve nota " + nota);
19 </script>
```

Mesmo que não seja tão trabalhoso assim, quando você tiver digitando pode se atrapalhar e esquecer um '+' para concatenar alguma palavra com a variável, então o JavaScript criou o **Template String**, que basicamente te dá a oportunidade de concatenar as variáveis e os caracteres dentro de CRASES `` , veja como ficaria o mesmo código apresentado acima com o **Template String**:

```
11 <script type="text/javascript">
12
13     var nome = "Felipe"
14     var idade = 27
15     var nota = 6.5
16     var matricula = 8754
17
18     alert(`O aluno ${nome} tem ${idade} anos de matrícula ${matricula} teve nota ${nota}`);
19 </script>
```

Depende muito do programador em encontrar a forma que achar mais confortável, as duas formas funcionam muito bem. Note que tudo que foi digitado no **alert()** está entre as crases ``.

Operadores

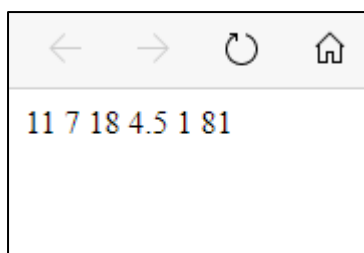
O JavaScript é uma linguagem com variáveis que não são fortemente tipadas. Isso significa que, embora as variáveis possuam tipos, eles só são atribuídos em tempo de execução. Por isso, não há definição explícita do tipo de dado: tudo é feito utilizando o termo "var". Por exemplo, um dado do tipo inteiro é definido como `var i = 0;`, enquanto um dado do tipo string é definido como `var s = 'texto'.`

Aritméticos

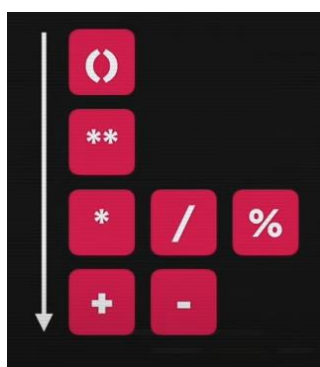
Servem para as operações matemáticas básicas, adição(+), subtração(-), multiplicação(*), divisão(/), resto(%) e potência(**). O código a seguir apresenta como esses operadores podem ser utilizados.

```
11 <script type="text/javascript">
12     soma      = 9+2    //operador de adição '+' Resultado 11
13     subtrai   = 9-2    //operador de subtração '-' Resultado 7
14     multiplicar = 9*2   //operador de multiplicador '*' Resultado 18
15     dividir   = 9/2    //operador de divisão '/' Resultado 4,5
16     resto     = 9%2    //operador de resto '%' Resultado 1
17     potencia  = 9**2   //operador de potência '**' Resultado 81
18     document.write(`${soma} ${subtrai} ${multiplicar} ${dividir} ${resto} ${potencia}`)
19 </script>
```

Resultado na página HTML:



Para os operadores aritméticos apresentados existe, assim como na matemática, a **ordem de precedência**, signica que existem operadores que possuem prioridade em relação a outros. Na imagem a seguir é apresentado na ordem mostrando a maior prioridade da parte superior até o inferior.



Tudo que tiver entre parenteses é calculado primeiro; em seguida vem a potência; depois a multiplicação, divisão e resto na mesma linha de prioridade, ou seja, se eles estiverem em uma mesma operação terão o mesmo peso e o código vai ler da esquerda para a direita (a famosa frase “a ordem dos fatores não altera o produto”); e por fim a adição e subtração.

Atribuição

Atribuição simples

Na atribuição simples, uma variável recebe o valor a partir do sinal igual (=).

```

11  <script type="text/javascript">
12      var a = 1 + 5      //Resultado: 6
13      var b = a + 2      //Resultado: 6(a) + 2 → 8
14      var c = b + a      //Resultado: 8(b) + 6(a) → 14
15      var d = 9 - a / 3   //Resultado: 9 - 6(a) / 3 → 9 - 2 → 7
16      var e = b % a + 2 * c //Resultado: 8 % 6 + 2 * 14 → 2 + 28 → 30
17
18      /*Note que no cálculo da variável 'd' primeiro é calculado a divisão e depois a subtração
19      já na variável 'e' primeiro faz o resto e a multiplicação da esquerda para a direita e em
20      seguida a adição*/
21  </script>

```

No exemplo acima as variáveis a, b, c, d ,e recebem os valores 6, 8, 14, 7 e 30 respectivamente. Digite o código em seu site e faça com que o código insira a variável **e** na tela.

Autoatribuição

As variáveis são peças importantes nos códigos, com elas que o programador vai manipular os dados, elas podem também receber o valor que já possuem e somar com alguma valor novo, veja no exemplo:

```
22      var a = 1 + 5      //Resultado: 6
23      a = a + 3          //Resultado: 6(a) + 3 → 9
24      a = a * 2          //Resultado: 9(a) * 2 → 18
25      a = a % 2          //Resultado: 18 % 2 → 0 (resto)
26
27      /*Note que o valor de 'a' é atualizado a cada autoatribuição, começou com uma
28      soma de valor 6, depois atualizou para 9, em seguida 18 e por fim 0.*/
```

Podemos também simplificar a autoatribuição utilizando os operadores +=, -=, *=, /=, %= e **=. Veja no exemplo:

```
30      var a = 1 + 5      //Resultado: 6
31      a += 3              //Resultado: 6(a) + 3 → 9
32      a *= 2              //Resultado: 9(a) * 2 → 18
33      a %= 2              //Resultado: 18 % 2 → 0 (resto)
34
35      /*Note que podemos simplificar a autoatribuição os operadores recebendo o sinal
36      aritmético junto com o igual (=), essa forma de representação funciona igual o exemplo
37      anterior. OBS: Só serve quando você realiza uma autoatribuição da mesma variável, Se a
38      variável 'a' do exemplo anterior receber o valor de uma variável distinta deve apresentar
39      ela na fórmula*/
40
41      var a = 1 + 5
42      var b = 2 - 1
43      a += 3
44      a = b + 6
```

Incremento

Incrementar significa aumentar o valor de algo, e decrementar significa diminuir o valor de algo (uma variável, por exemplo). Neste caso, o operador de incremento ++ aumenta o valor de uma variável em 1 (sempre 1), e o operador de decremento -- diminui o valor da variável em 1 (sempre 1).

```
48      /* Incremento */
49
50      var num = 5          //Variável 'num' recebe o valor 5
51      num++                //É incrementado (somado) na variável o valor 1
52      alert("1º Incremento → " + num) //Mostra o valor 6
53      num++                //É incrementado (somado) na variável o valor 1
54      alert("2º Incremento → " + num) //Mostra o valor 7
55      num--                //É decrementado (subtraído) na variável o valor 1
56      alert("1º Decremento → " + num) //Mostra o valor 6
```

Essa técnica de incremento é muito utilizado por programadores para delimitar repetições de determinadas operações. Ao passo que o código é solicitado vai se incrementando até chegar ao limite definido. No capítulo de repetições você verá isso com mais precisão.

Relacionais

Os operadores **relacionais**, assim como os de igualdade, avaliam dois operandos. Neste caso, mais precisamente, definem se o operando à esquerda é menor, menor ou igual,

maior, maior ou igual e diferente ao da direita, retornando um valor booleano (verdadeiro ou falso).

A tabela abaixo apresenta os símbolos de cada operador relacioanal.

Símbolo	Nome do Operador	Exemplo	Significado
>	Maior que	$x > y$	x é maior que y?
>=	Maior ou igual	$x \geq y$	x é maior ou igual a y ?
<	Menor que	$x < y$	x é menor que y?
<=	Menor ou igual	$x \leq y$	x é menor ou igual a y ?
==	Igualdade	$x == y$	x é igual a y?
!=	Diferente de	$x != y$	x é diferente de y?

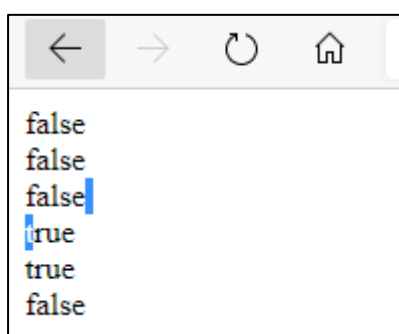
O código abaixo apresenta como podemor utilizar os operadores para realizar essa verificação.

```

11 <script type="text/javascript">
12     var a = 10 //Declarando a variável a
13     var b = 10 //Declarando a variável b
14     document.write((a > b) + "<br/>") // 'a' é maior que 'b'? False
15     document.write((a < b) + "<br/>") // 'a' é menor que 'b'? False
16     document.write((a >= 20) + "<br/>") // 'a' é maior ou igual a 20? False
17     document.write((a <= 10) + "<br/>") // 'a' é menor ou igual a 10? True
18     document.write((a == b) + "<br/>") // 'a' é igual a 'b'? True
19     document.write((a != b) + "<br/>") // 'a' é diferente de 'b'? False
20
21     /* OBS: A tag <br/> que é do HTML pode ser utilizada aqui para dar uma quebra
22     linha assim como se faz no HTML. Note que para que eu pudesse utiliza-lo, tive
23     que colocar a minha operação entre parenteses e concatenei com o '+' a tag <BR>*/
24 </script>

```

Veja o resultado no navegador:



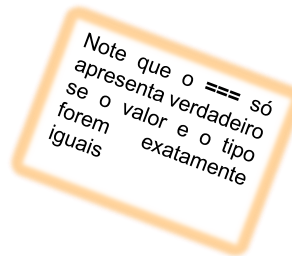
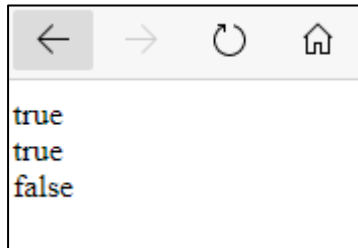
Perceba as quebras
de linhas entre os
resultados

Identidade

Temos ainda mais um operador de identidade (===), diferente do igual a (==), o de identidade verifica se os valores comparados são exatamente iguais, tanto no valor quanto no tipo, enquanto que o igual a só compara o valor e não o tipo. Veja no exemplo:


```
25 var a = 10 //Declarando a variável a
26 document.write((a == 10) + "<br/>") // 'a' é igual ao número 10? true
27 document.write((a == '10') + "<br/>") // 'a' é igual a string 10? true
28 document.write((a === '10') + "<br/>") // 'a' é semelhante a string 10? false
```

Resultado no navegador:

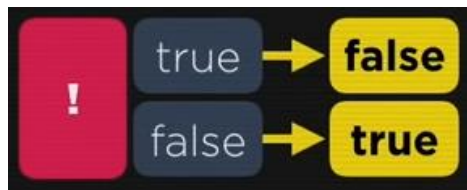


Lógicos

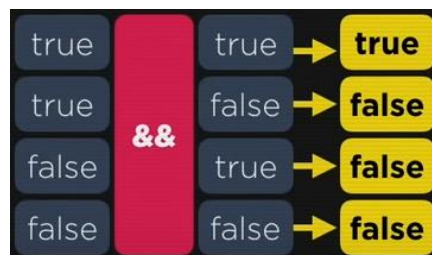
Os operadores lógicos são representados pela ! (negação), os && (conjunção) e o || (disjunção).



- Negação
 - É o mais simples dos três e basicamente nega uma informação entregue, por exemplo: Felipe pede emprestado uma caneta para João e indica que **NÃO** pode ser vermelha, qualquer outra caneta vai satisfazer Felipe.

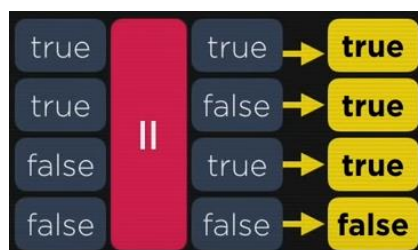


- Conjunção
 - Neste temos o **E** lógico, onde a situação será satisfeita se as duas situações sejam contempladas por completo, por exemplo: Felipe pede para José uma caneta preta **E** uma caneta verde, então para satisfazer Felipe só servem as duas das cores solicitadas



- Disjunção

- Neste temos o **OU** lógico, onde a situação é satisfeita se apenas uma das situações seja contemplada, por exemplo: Felipe pede para Maria uma caneta azul **OU** uma caneta preta, então basta uma das duas ou as duas para satisfazer Felipe.



Vamos fazer um exemplo com você, vou criar 3 exemplos e você responde se é verdade ou falso para.

- idade \geq 18 **&&** idade \leq 50? Leia assim: a idade é maior ou igual a 18 **E** menor ou igual a 50?
- estado == 'BA' **||** estado == 'RO'? Leia assim: o estado é Bahia **OU** Rondônia?
- sexo != "M" **&&** altura > 160cm? Leia assim: **NÃO** é homem **E** tem mais de 160 cm de altura?

E aí? No meu ficou **Verdadeiro** para a 1ª pergunta pois tenho 27 anos, **Verdadeiro** na 2ª pois sou da Bahia e no 3º mesmo que eu tenha mais de 160 cm de altura, eu sou homem e a proposição fica **Falsa**.

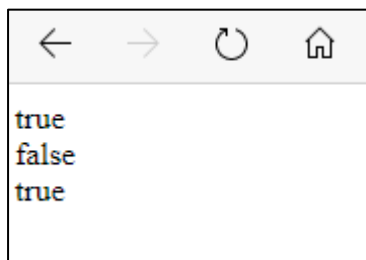
Veja agora no código como podemos fazer essas operações:

```

12  var nome = 'João'
13  var estado = 'Acre'
14  var idade = 30
15  var altura = 182 //Em centímetros
16  var sexo = "M"
17
18  //Compara se o nome é José 'OU' se o estado é o Acre
19  document.write(nome == 'José' || estado == 'Acre')
20  document.write("<br/>")
21
22  //Compara se a idade é maior ou igual a 18 'E' altura menor que 150 centímetros
23  document.write(idade >= 18 && altura < 150)
24  document.write("<br/>")
25
26  //Compara se o sexo 'não' é feminino 'E' se o estado 'não' é a Bahia
27  document.write(sexo != "F" && estado != "Bahia")
28  document.write("<br/>")

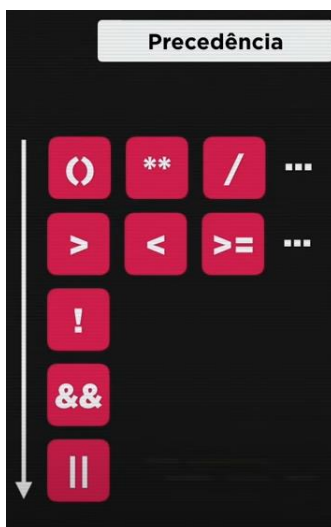
```

Resultado:



```
true
false
true
```

Nossa ordem de precedência se atualizou após todos esses conteúdos de operadores, em uma expressão a ordem de prioridade é a seguinte:



Blocos Condicionais

Em qualquer linguagem de programação, o código precisa tomar decisões e realizar ações de acordo, dependendo de diferentes entradas. Por exemplo, em um jogo, **se** o número de vidas do jogador é 0, **então** o jogo acaba. Em um aplicativo de clima, **se** estiver sendo observado pela manhã, **então** ele mostra um gráfico do nascer do sol; mostra estrelas e uma lua se for noite.

Seres humanos (e outros animais) tomam decisões o tempo todo que afetam suas vidas, desde pequenas ("devo comer um biscoito ou dois?") até grandes ("devo ficar no meu país de origem e trabalhar na fazenda do meu pai ou devo mudar para a América e estudar astrofísica?").

As declarações condicionais nos permitem representar tomadas de decisão como estas em JavaScript, a partir da escolha que deve ser feita (por exemplo, "um biscoito ou dois"), ao resultado obtido dessas escolhas (talvez o resultado de "comer um biscoito" possa ser "ainda sentido fome", e o resultado de "comer dois biscoitos" pode ser "ter se sentido cheio, mas mamãe me falou para comer todos os biscoitos").

De longe o tipo mais comum de declaração condicional que você usará em JavaScript as modestas declarações **if .. else (Se .. Então)**.

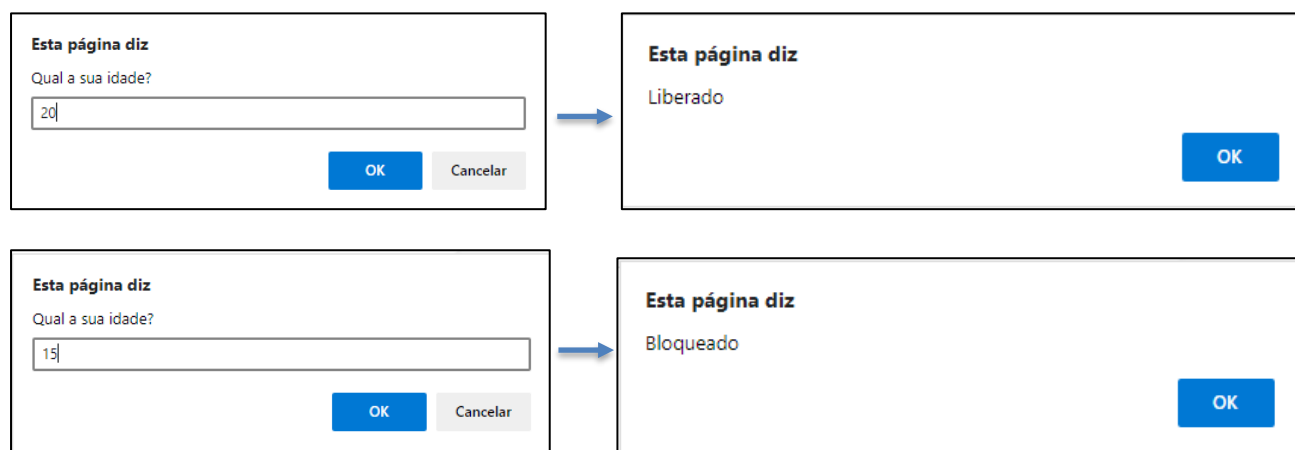
Veja a sintaxe básica do **if...else** no pseudocódigo:

```
if (condicao) {  
    código para executar caso a condição seja verdadeira  
} else {  
    código para executar caso a condição seja falsa  
}
```

Veja o código a seguir. As condições que foram abordadas no capítulo anterior são utilizadas nessas condições da condição IF ELSE.

```
11 <script type="text/javascript">  
12     var idade = prompt("Qual a sua idade?")  
13  
14     if (idade >= 18){           //Verifica se a idade digitada é maior ou igual a 18  
15         alert("Liberado")      //Se for maior que 18 o programa entra nesse bloco e apresenta a mensagem 'liberado'  
16     }else{                     //Senão, ou seja, caso não seja maior ou igual a 18 anos  
17         alert("Bloqueado")     //Entra nesse bloco e apresenta a mensagem 'Bloqueado'  
18     }  
19 </script>
```

Resultados:



Funções

As funções são blocos de comandos com instruções pré-definidas que executam uma tarefa ou calculam um valor. Imagine a função como um departamento de uma empresa, quando o chefe precisa de algum relatório financeiro, ele entrega os valores ao departamento e recebe esse relatório, se ele precisa de um estudo sobre os funcionários ele entrega os dados ao RH e eles retornam o relatório.

O Chefe dessa empresa não precisa contratar novas pessoas para fazer o relatório, uma vez que ele já tem o departamento em sua empresa, e assim é em programação, qualquer conjunto de instruções pode virar uma função (departamento) e será solicitado apenas quando for necessário.

As funções são ocorridas apenas se algo acontecer, seja se o usuário clicar em algum local, seja quando o usuário digita um valor etc. A definição da função (também chamada de declaração de função) consiste no uso da palavra-chave **function**, seguida por:

- Nome da Função.
- Lista de argumentos para a função, entre parênteses e separados por vírgulas.
- Declarações JavaScript que definem a função, entre chaves { }.
-

Por exemplo, o código a seguir define uma função simples chamada square:

```
13 function square(numero) {  
14     return numero * numero;  
15 }  
16
```

A função **square** recebe um argumento chamado **numero**. A função consiste em uma instrução que indica para retornar o argumento da função (isto é, numero) multiplicado por si mesmo. A declaração **return** especifica o valor retornado pela função. Neste caso, se o número recebido fosse 5 o valor retornado (no return) seria 25.

Sempre que estivermos programando em qualquer linguagem, temos que ter em mente uma coisa muito importante: "o melhor jeito de resolver um problema grande é dividi-lo em problemas menores", cada um desses "problemas menores" serão resolvidos por funções pequenas, assim juntando as várias pequenas funções, teremos resolvido "o todo".

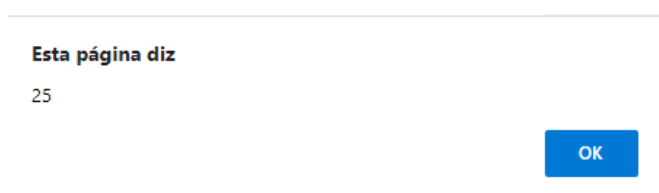
Apesar de ser possível escrever todo o código que resolve o problema grande numa única função gigante, não fazemos isso. Pois isso tornaria muito complexo nosso código, dificultaria uma futura manutenção e impossibilitaria o reaproveitamento de pequenas rotinas. Por isso preferimos dividir e depois criar uma função grande que utilize nossas outras funções pequenas, do que escrever tudo num só lugar.

INVOCANDO

Após construirmos a função, ela por si só não faz absolutamente nada até a chamarmos. A invocação consiste em colocar o nome da função seguido pelos parênteses. Isso faz com que o código dentro do corpo da nossa função seja executado.

```
14 function quadrado(numero) {  
15     return alert(numero * numero);  
16 }  
17  
18 quadrado(5) //chamando a função quadrado passando como parâmetro o valor 5  
19  
20 /* desta forma a função vai calcular o quadrado de 5 e apresentar na tela */
```

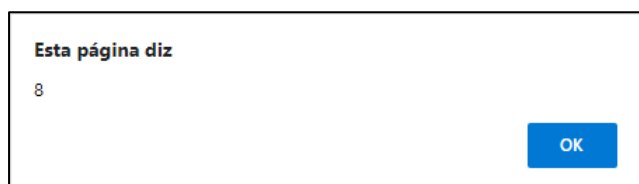
Resultado:



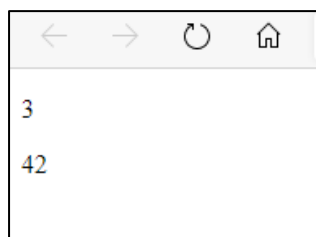
Veja este outro exemplo onde temos uma função `somar()` recebendo dois parâmetros e retornando a soma dos dois. Na linha 21 o código exibe uma mensagem com a invocação da função `somar()`. Na linha 25 é inserido no **parágrafo** de id **resultado** o valor da função `somar(1, 2)` e por fim na linha 29 é inserido no **parágrafo** de id **resultado2** o valor da função `somar(10, 32)`.

```
8 <body>
9
10 <p id="resultado"></p>
11 <p id="resultado2"></p>
12
13 <script>
14
15
16     function somar(x, y) {
17         return x + y;
18     }
19
20     //invocando a função somar() passando como parâmetros 3 e 5
21     alert(somar(3, 5));
22
23     //Invocando a função somar() passando como parametros 1 e 2 e inserindo o resultado
24     //no parágrafo de id 'resultado'
25     document.getElementById('resultado').innerHTML = somar(1, 2);
26
27     //Invocando a função somar() passando como parametros 10 e 32 e inserindo o resultado
28     //no parágrafo de id 'resultado2'
29     document.getElementById('resultado2').innerHTML = somar(10, 32);
30
31 </script>
32 </body>
```

Resultado:



Resultado do alert
da linha 21



Resultado
linhas 25 e 29 das

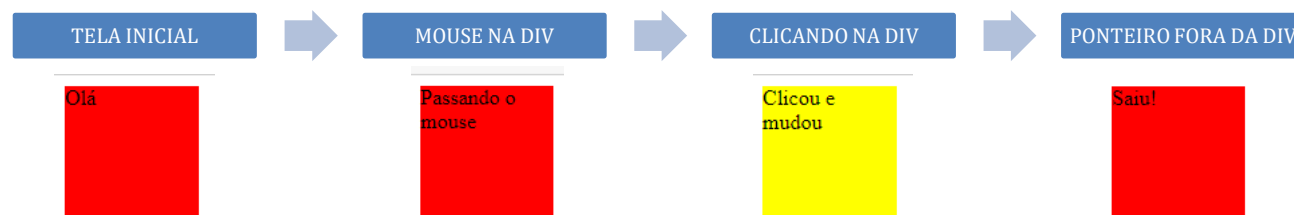
Eventos

Os eventos são basicamente um conjunto de ações que são realizadas em um determinado elemento da página web, seja ele um texto, uma imagem, ou uma div, por exemplo. Muitas das interações do usuário que está visitando sua página com o conteúdo do seu site podem ser consideradas eventos.

Existe uma infinidade de eventos definidos para uso em JavaScript e no exemplo a seguir mostro 3 deles que são o **click**, **mouseenter** e **mouseout**.

- Click
 - Esse evento é iniciado quando o usuário clica em alguma parte específica, no caso do exemplo abaixo esse evento era iniciado quando clicava na Div com o texto **Olá** criado no HTML
- Mouseenter
 - Esse evento é iniciado quando o mouse passa por algum local específico, no exemplo abaixo bastava o usuário mover o ponteiro do mouse pela Div do HTML.
- Mouseout
 - Esse evento é iniciado quando é retirado de alguma área específica, no exemplo abaixo o usuário retirando o mouse da Div iniciava o evento.

```
8 <style>
9   div#exemplo { /* formatação para colocar tamanho 100x110 e cor vermelha */
10     width: 100px;
11     height: 100px;
12     background: red;
13   }
14 </style>
15 </head>
16 <body>
17   <div id="exemplo"> <!-- Div que será modificada a depender do evento que for iniciado -->
18     Olá
19   </div>
20
21   <script>
22     var a = window.document.getElementById('exemplo') //Pega o a Div para modificar
23
24     //Fica em Stand by "escutando" o movimento do mouse, a depender do que aconteça ele chama a função específica
25     a.addEventListener('click', clicar)
26     a.addEventListener('mouseenter', entrar)
27     a.addEventListener('mouseout', sair)
28
29     function clicar(){ //Se o usuário clicar na div o código chama essa função
30       a.innerText = 'Clicou e mudou' //Muda o texto que estiver na Div para 'Clicou e mudou'
31       a.style.background = "yellow" //Altera a cor da Div para amarelo
32     }
33
34     function entrar(){ //Se o usuário passar o mouse sobre a Div, o código chama essa função
35       a.innerText = "Passando o mouse" //Mostra a mensagem quando o usuário movimentar o ponteiro do mouse na div
36     }
37
38     function sair(){ //Se o usuário retirar o ponteiro da Div, o código chama essa função
39       a.innerText="Saiu!" //Mostra a mensagem 'Saiu' quando o usuário retirar o ponteiro da Div
40       a.style.background="red" //Muda a cor para vermelho quando o ponteiro sair
41     }
42   </script>
```



Existem muitos eventos, você pode encontrá-los no site de Referência do Mozilla, acesse clicando [AQUI](#).

Bloco de repetições

Uma atividade que os computadores realizam com efetividade é executar algo várias vezes. Desde que seja programado corretamente. Felizmente, para não precisamos repetir inúmeras vezes a invocação de uma função ou certo código, existe os loops (laços de repetição). Há três tipos de estruturas de repetição:

FOR

Tem como objetivo repetir um bloco de código um determinado número de vezes até sua condição final ser atingida.

Sintaxe:

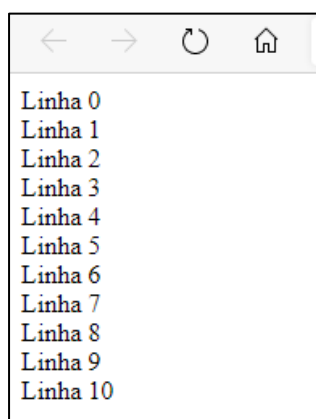
```
<script>  
  for (condicaoInicial; condicaoFinal; incremento) {  
    executa bloco de código;  
  }  
</script>
```

Quando um for é executado, ocorre o seguinte:

- A expressão **condicaoInicial** é inicializada e, caso possível, é executada. Normalmente essa expressão inicializa um ou mais contadores, mas a sintaxe permite expressões de qualquer grau de complexidade. Podendo conter também declaração de variáveis.
- A expressão **condicaoFinal** é avaliada. caso o resultado de **condicaoFinal** seja verdadeiro, o laço é executado. Se o valor de **condicaoFinal** é falso, então o laço terminará. Se a expressão **condicaoFinal** é omitida, a **condicaoFinal** é assumida como verdadeira.
- A instrução é executada. Para executar múltiplas declarações, use uma declaração em bloco ({ ... }) para agrupá-las.
- A atualização da expressão **incremento**, se houver, executa, e retorna o controle para o passo 2.

```
10 <script>  
11 //Ele começa com 'i = 0', e a cada laço pergunta para o for se 'i <= 10', caso seja ele entra no bloco  
12 //e executa as suas intruções, a cada entrada no bloco é incrementado o valor 1 à variável 'i'.  
13 for (var i=0; i<= 10; i++) { // Condição incial | Condição de parada | Incremento  
14   document.write('Linha ' + i + "<br/>"); //Escreve na página o texto com o valor de i naquele momento  
15 }  
16 </script>
```

Resultado:



Linha 0
Linha 1
Linha 2
Linha 3
Linha 4
Linha 5
Linha 6
Linha 7
Linha 8
Linha 9
Linha 10

Veja como em apenas duas linhas de código o programa realizou 10 repetições e apresentou no site

WHILE (Enquanto)

Executa seu bloco de instruções **enquanto** a sua condição seja verdadeira.

Sintaxe:

```
<script>
  while (condicao) {
    bloco de operação
  }
</script>
```

Se a condição se tornar falsa, a declaração dentro do laço (**condicao**) para a execução e o controle é passado para a instrução após o laço.

O teste da condição ocorre antes que o laço seja executado. Desta forma se a condição for verdadeira o laço executará e testará a condição novamente. Se a condição for falsa o laço termina e passa o controle para as instruções após o laço.

```
12 <script>
13   //inicializo a variável com o valor 1 (A primeira fatia da pizza)
14   var comerFatia = 1
15
16   //A condição while(enquanto) tem como condição de entrada o valor em comerFatia
17   //ser menor ou igual a 8, quando ficar igual a 8 ou ultrapassar o laço não será utilizado
18   while (comerFatia <= 8) {
19
20       //Insere amensagem no site com o valor da variável Fatia
21       document.write('comer ' + comerFatia + " fatia <br/>");
22
23       //Incrementa o valor 1 na variável comerFatia
24       comerFatia++;
25   }
26 </script>
```

Resultado:



```
comer 1 fatia
comer 2 fatia
comer 3 fatia
comer 4 fatia
comer 5 fatia
comer 6 fatia
comer 7 fatia
comer 8 fatia
```

Existem outros tipos de laços, recomendo acessar o site de referência do Mozilla através [DESTE LINK](#) e ver os outros métodos possíveis. Porém, o **For** e o **While** são os mais utilizados.

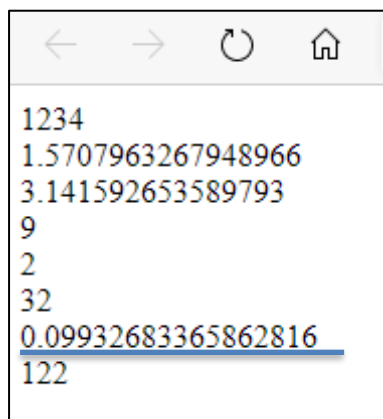
Objeto Math

Math em inglês significa matemática. O objeto Math tem propriedades e métodos para constantes matemáticas e funções. Por exemplo, o **PI** do objeto **Math** tem o valor de pi (3,141...), que você usaria em uma aplicação como

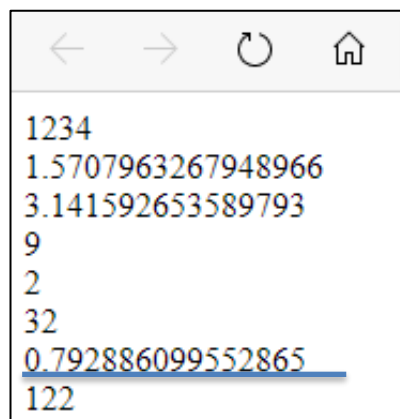
Similarmente, funções matemáticas padrão são métodos do **Math**. Isto inclui funções trigonométricas, logarítmicas, exponenciais, e outras funções. Por exemplo, se você quiser usar a função trigonométrica seno, basta escrever **Math.sin (1.56)**.

```
11 <script>
12     document.write(Math.abs(-1234) + "<br/>") //Módulo de um número
13     document.write(Math.acos(0,4) + "<br/>") //Arco co-seno (em radianos) de um número
14     document.write(Math.PI + "<br/>") //Pi
15     document.write(Math.max(2,9) + "<br/>") //O maior valor entre os números
16     document.write(Math.min(2,9) + "<br/>") //Menor valor entre os números
17     document.write(Math.pow(2,5) + "<br/>") //Potencia onde o primeiro é a base e o segundo a potência
18     document.write(Math.random() + "<br/>") //Número aleatório entre 0 e 1 com até 15 dígitos
19     document.write(Math.round(121.6) + "<br/>")//Arredonda o número
20 </script>
```

Resultado:



1234
1.5707963267948966
3.141592653589793
9
2
32
0.09932683365862816
122



1234
1.5707963267948966
3.141592653589793
9
2
32
0.792886099552865
122

A cada atualização na página um novo valor é apresentado para o **random**.

Data e Hora

JavaScript não possui dados do tipo data. No entanto, você pode usar o objeto **Date** e seus métodos para trabalhar com datas e horas nas suas aplicações. O objeto **Date** tem um grande número de métodos para setar, recuperar e manipular datas. Ele não tem nenhuma propriedade.

JavaScript manipula datas de maneira semelhante ao Java. As duas linguagens tem muitos dos mesmos métodos para lidar com datas e ambas armazenam datas como números em milissegundos, desde 1 de janeiro de 1970, às 00:00:00 (January 1, 1970, 00:00:00).

A abrangência do objeto **Date** é de -100,000,000 dias até 100,000,000 dias relativos a 01 de janeiro de 1970 UTC.

Note a pegadinha do mês, que inicia com zero e não um. Se você não somar 1, janeiro aparecerá como 0 e dezembro como 11. Já o dia do mês vai de 1 a 31, não é preciso somá-lo. O dia da semana também inicia em zero, representando o domingo, e vai até seis (sábado). Use um **array** para mostrar o dia da semana por extenso:

```

11 <script>
12 // Obtém a data/hora atual
13 var data = new Date();
14
15 // Guarda cada pedaço em uma variável
16 var dia = data.getDate(); // 1-31
17 var dia_sem = data.getDay(); // 0-6 (zero=domingo)
18 var mes = data.getMonth(); // 0-11 (zero=janeiro)
19 var ano2 = data.getFullYear(); // 2 dígitos
20 var ano4 = data.getFullYear(); // 4 dígitos
21 var hora = data.getHours(); // 0-23
22 var min = data.getMinutes(); // 0-59
23 var seg = data.getSeconds(); // 0-59
24 var mseg = data.getMilliseconds(); // 0-999
25 var tz = data.getTimezoneOffset(); // em minutos
26
27 // Formata a data e a hora (note o mês + 1)
28 var str_data = dia + '/' + (mes+1) + '/' + ano4;
29 var str_hora = hora + ':' + min + ':' + seg;
30
31 var dias = new Array('domingo', 'segunda', 'terça', 'quarta', 'quinta', 'sexta', 'sábado');
32
33 //Mostra o resultado do dia com o nome baseado no array
34 alert('Hoje é ' + dias[data.getDay()] + ", dia " + str_data + ' às ' + str_hora);
35
36 </script>

```

Bônus:

Array

Com o Array é possível armazenar um conjunto de quaisquer valores JavaScript, como números, caracteres ou textos ou uma mistura deles. Imagine o array como um gaveteiro onde você pode adicionar ou retirar gavetas e cada gaveta contém o objeto que quiser, vamos criar aqui um gaveteiro onde a primeira gaveta contém o valor 10 a segunda 20 e a terceira 30.

```

11 <script>
12     var gaveteiro = [10, 20, 30];
13 </script>

```

Acessando elementos do array

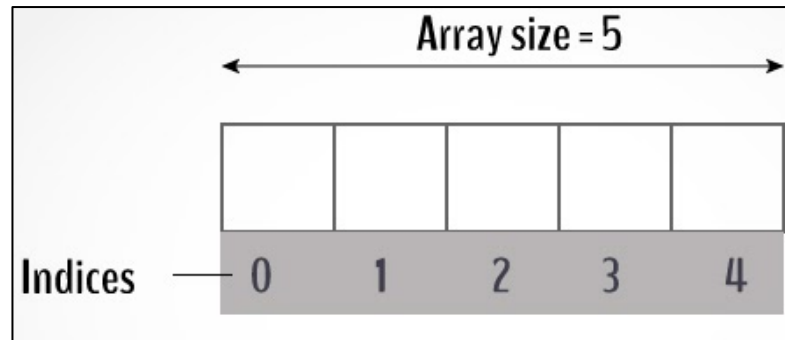
Um Array é uma variável especial, que pode conter mais de um valor por vez. Com o array criado podemos visualizar cada uma das posições individualmente colocando o índice dentro de colchetes:

```

11 <script>
12     var gaveteiro = [10, 20, 30]
13
14     document.write(gaveteiro[2] + "<br/>") //30
15     document.write(gaveteiro[1] + "<br/>") //20
16     document.write(gaveteiro[0] + "<br/>") //10
17
18 </script>

```

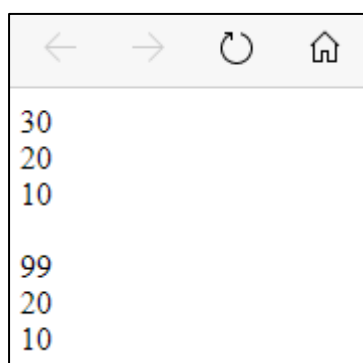
OBS: Repare que para acessar os índices é necessário colocar o número do item desejado.



Note que o índice começa com o valor 0 para o primeiro item. Também podemos alterar o valor de cada posição da seguinte forma:

```
11 <script>
12   var gaveteiro = [10, 20, 30]
13
14   document.write(gaveteiro[2] + "<br/>") //30
15   document.write(gaveteiro[1] + "<br/>") //20
16   document.write(gaveteiro[0] + "<br/>") //10
17
18   gaveteiro[2] = 99;
19   document.write("<br/>");
20
21   document.write(gaveteiro[2] + "<br/>") //?
22   document.write(gaveteiro[1] + "<br/>") //?
23   document.write(gaveteiro[0] + "<br/>") //?
24
25
26 </script>
```

Resultado:



O índice 2 foi alterado e recebeu o valor 99, sendo assim o valor 30 que antes estava atribuído não existe mais.