APOSTILA

JavaScript

Material Desenvolvido pela

w3schools.com

Saiba mais em https://www.w3schools.com/

Adaptado e traduzido para o Português

JavaScript é a principal linguagem de script da Web e é utilizado em bilhões de páginas da Web para adicionar funcionalidade, validar formulários, comunicar com o servidor, e muito mais.

Esta apostila foi baseada no material da W3Schools e traduzida com adequações. O material original (inglês) e na íntegra pode ser encontrado na URL: http://www.w3schools.com/js/

ÍNDICE

1. Introdução 7
1.1. O que é JavaScript? 7
1.2. O que você aprenderá 7
1.3. JavaScript: Escrevendo na saída HTML 7
1.4. JavaScript: reagir aos acontecimentos 7
1.5. JavaScript: Alterando conteúdo HTML 8
1.6. JavaScript: Alterando Imagens HTML 8
1.7. JavaScript: Alterando Estilos HTML 9
1.8. JavaScript: validar a entrada 9
1.9. O que mais posso fazer com o JavaScript? 9

2. Output utilizando JavaScript 11
2.1. Manipular elementos HTML 11
2.2. Escrevendo para a saída de documentos 11
2.3. Aviso 12
2.4. Console 12

3. Funções 13
3.1. Sintaxe da função JavaScript 13
3.2. Chamar uma função com argumentos 13
3.3. Funções com um valor de retorno 14
3.4. Variáveis locais 15
3.5. Variáveis globais 15
3.6. O tempo de vida das variáveis 16
3.7. Atribuir valores às variáveis de JavaScript não declarados 16

4. Onde inserir o código Javascript 17
4.1. JavaScript em <body> 17
4.2. Funções JavaScript e Eventos 17
4.3. Funções de JavaScript em <head> 17
4.4. Funções de JavaScript em <body> 18
4.5. Scripts em <head> e <body> 18
4.6. Usando um JavaScript externo 18

5. Operadores Javascript 20
5.1. Operadores aritméticos JavaScript 20
5.2. Operadores de atribuição JavaScript 20
5.3. O Operador + usado em strings 20
5.4. Concatenando Strings e Números 21

6. Operadores Lógicos e de Comparação 22
6.1. Operadores de Comparação 22
6.2. Como ele pode ser usado 22
6.3. Operadores lógicos 22
6.4. Operador condicional 23

7. If ... Else 24
7.1. Declarações Condicionais 24

8. Switch 25
8.1. A instrução switch JavaScript 25
8.2. O caso “default” 26

9. Laços 27
9.1. Laços em Javascript 27
9.2. O laço FOR 27
9.3. O laço WHILE (verifica condição no início do bloco) 28
9.4. O laço DO ... WHILE (verifica a condição no final do bloco) 28
9.5. Comparando for e while 29

10. Primeiros Programas 30
10.1. Demonstrações JavaScript 30
10.2. Código JavaScript 30
10.3. Blocos de JavaScript 30
10.4. JavaScript é sensível a maiúsculas 31
10.5. Espaço em Branco 31
10.6. Quebrar uma linha de código 31

11. Comentários 32
11.1. Comentário Multilinha 32
11.2. Usando comentários para impedir a execução 32
11.3. Usando comentários no final de uma linha 32

12. Variáveis 33
12.1. Você se lembra de Álgebra? 33
12.2. Variáveis em JavaScript 33
12.3. Tipos de dados JavaScript (introdução) 33
12.4. Declarando (Criação) Variáveis em JavaScript 34
12.5. Variáveis locais JavaScript 34
12.6. Variáveis globais JavaScript 35
12.7. Atribuindo valores a variáveis não declaradas JavaScript 35
12.8. Uma declaração, muitas variáveis 35
12.9. Valor = indefinido 35
12.10. Re-Declarando Variáveis JavaScript 35
12.11. Aritmética JavaScript 36
12.12. Acessando o conteúdo de INPUTS 36

13. Tipos de Dados 37
13.1. JavaScript Tem tipos dinâmicos 37
13.2. String 37
13.3. Números 37
13.4. Booleanos 37
13.5. Arrays 38
13.6. Objetos 38
13.7. Indefinido e Null 38
13.8. Declarando Tipos de variáveis 39

14. Objetos em JavaScript 40
14.1. Propriedades e métodos 40
14.2. Um objeto da Vida Real: um carro 40
14.3. Objetos em JavaScript: 40
14.4. Criação de objetos JavaScript 41
14.5. Acessando propriedades de objetos 41
14.6. Acessando Métodos de Objeto 41

15. Caixas de Diálogo 42
15.1. Caixa de Alerta 42
15.2. Caixa de Confirmação 42
15.3. Caixa de Prompt (Pergunta) 43
15.4. Quebras de linha 43

16. Break e Continue 45
16.1. A instrução break 45
16.2. A instrução continue 45

17. Tratando Erros: Try, Catch e Throw 46
17.1. Erros vão acontecer! 46
17.2. JavaScript Lança Erros 46
17.3. JavaScript Tentar Pegar os Erros 46
17.4. A instrução throw 47

18. Validação de Formulário 49
18.1. Campos obrigatórios 49
18.2. E-mail de validação 49

19. DOM (Document Object Model) 51
19.1. Encontrar elementos HTML 51
19.2. Encontrar elementos HTML por Id 51
19.3. Encontrar elementos HTML por Tag 52

20. DOM: Mudando o HTML 53
20.1. Alterando o fluxo de saída HTML 53
20.2. Alterar conteúdo HTML 53
20.3. Mudando um atributo HTML 54

21. DOM: Mudando Propriedades CSS 55
21.1. Mudar o estilo HTML 55

22. DOM: Eventos 56
22.1. Reagir aos acontecimentos 56
22.2. Atributos de eventos HTML 56
22.3. Atribuir eventos usando o DOM HTML 57
22.4. Os eventos onload e onunload 57
22.5. O evento onchange 57
22.6. Os eventos onmouseover e onmouseout 57
22.7. Os eventos onmousedown, onMouseUp e onclick 58

23. DOM: adicionando e removendo nós (Elementos) 61
23.1. Criação de novos elementos HTML 61
23.2. Remoção de elementos HTML existente 61

1. Introdução

JavaScript é a linguagem de programação mais popular do mundo. É a linguagem de HTML e web, para os servidores, PCs, laptops, tablets, smartphones e muito mais.

1.1. O que é JavaScript?

- O JavaScript foi desenvolvido para adicionar interatividade a páginas HTML;
- JavaScript é uma linguagem de script;
- A linguagem de script é uma linguagem de programação leve;
- JavaScript é normalmente embutido diretamente em páginas HTML;
- JavaScript é uma linguagem interpretada (significa que scripts são executados sem compilação preliminar);
- Todo mundo pode usar JavaScript sem comprar uma licença.
- JavaScript é fácil de aprender.

1.2. O que você aprenderá

Abaixo está uma amostra do que você vai aprender nessa apostila.

1.3. JavaScript: Escrevendo na saída HTML

```
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph</p>");
```

Você só pode usar document.write na saída HTML. Se você usá-lo depois que o documento foi carregado, todo o documento será substituído.

1.4. JavaScript: reagir aos acontecimentos

Exemplo

```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript</h1>

<p>
JavaScript can react to events. Like the click of a button:
</p>

<button type="button" onclick="alert('Welcome!')">Click
Me!</button>

</body>
</html>
```

A função alert() não é muito usado em JavaScript, mas muitas vezes é bastante útil para experimentar código.

O evento onclick é apenas um dos muitos eventos do HTML, você vai aprender.

1.5. JavaScript: Alterando conteúdo HTML

Usando JavaScript para manipular o teor de elementos de HTML é uma funcionalidade muito poderosa.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript</h1>

<p id="demo">
JavaScript can change the content of an HTML element.
</p>

<script>
function myFunction()
{
x=document.getElementById("demo");  // Find the element
x.innerHTML="Hello JavaScript!";    // Change the content
}
</script>

<button type="button" onclick="myFunction()">Click Me!</button>

</body>
</html>
```

Muitas vezes você vai ver `document.getElementById ("id")`. Isso é definido no HTML DOM. O DOM (*Document Object Model*) é um padrão W3C oficial para acessar elementos HTML.

1.6. JavaScript: Alterando Imagens HTML

Este exemplo altera dinamicamente a fonte atributo (src) de um elemento `<image>` HTML:

```
<!DOCTYPE html>
<html>
<body>
<script>
function changeImage()
{
element=document.getElementById('myimage')
if (element.src.match("bulbon"))
  {
  element.src="pic_bulboff.gif";
  }
else
  {
  element.src="pic_bulbon.gif";
  }
```

```
}
</script>

<img id="myimage" onclick="changeImage()"
src="pic_bulboff.gif" width="100" height="180">

<p>Click the light bulb to turn on/off the light</p>

</body>
</html>
```

JavaScript pode alterar a maioria dos atributos de qualquer elemento HTML, não apenas imagens.

1.7. JavaScript: Alterando Estilos HTML

Mudar o estilo de um elemento HTML, é uma variante de mudar um atributo HTML.

Exemplo

```
x=document.getElementById("demo")  //Find the element
x.style.color="#ff0000";           //Change the style
```

1.8. JavaScript: validar a entrada

JavaScript é comumente usado para validar a entrada.

Exemplo

```
if isNaN(x) {alert("Not Numeric")};
```

1.9. O que mais posso fazer com o JavaScript?

- **JavaScript dá aos designers uma ferramenta de programação HTML –** Autores de HTML normalmente não são programadores, mas JavaScript é uma linguagem de script com uma sintaxe muito simples! Quase qualquer um pode colocar pequenos "trechos" do código em suas páginas HTML.
- **JavaScript pode reagir a eventos -** O JavaScript pode ser configurado para executar quando algo acontece, como quando uma página tenha terminado de carregar ou quando um usuário clica em um elemento HTML;
- **JavaScript pode manipular elementos HTML -** O JavaScript pode ler e alterar o conteúdo de um elemento HTML;
- **JavaScript pode ser usado para validar os dados -** O JavaScript pode ser usado para validar a entrada de formulários;
- **JavaScript pode ser usado para detectar o navegador do visitante** - dependendo do browser do usuário pode carregar outra página especificamente projetada para o navegador dele;

- **JavaScript pode ser usado para criar cookies** - O JavaScript pode ser usado para armazenar e recuperar informações sobre o computador do visitante.

Java e JavaScript são duas linguagens completamente diferentes, tanto em conceito e design.
Java (inventado pela Sun) é uma linguagem de programação mais complexa na mesma categoria do C. ECMA-262 é o nome oficial do padrão JavaScript. JavaScript foi inventado por Brendan Eich. Ela apareceu em Netscape (o navegador já não existente), em 1995, e foi adotado pela ECMA (uma associação de padronização) desde 1997.
O padrão ECMA (chamado ECMAScript-262) foi aprovado como um padrão ISO internacional (ISO / IEC 16262) padrão em 1998. O desenvolvimento ainda está em andamento.

2. Output utilizando JavaScript

JavaScript é normalmente usado para manipular elementos HTML.

2.1. Manipular elementos HTML

Para acessar um elemento HTML JavaScript, você pode usar o método document.getElementById(id).

Use o atributo "id" para identificar o elemento HTML:

Exemplo: Acesse o elemento HTML com o especificado id e alterar o seu conteúdo

```
<!DOCTYPE html>
<html>
<body>

<h1>Minha primeira página web</h1>

<p id="demo">Meu primeiro parágrafo</p>

<script>
document.getElementById("demo").innerHTML="Meu primeiro
Javascript";
</script>

</body>
</html>
```

O JavaScript é executado pelo navegador da web. Neste caso, o navegador irá acessar o elemento HTML com o `id = "demo"`, e substituir seu conteúdo (`innerHTML`) com "*My First JavaScript*".

2.2. Escrevendo para a saída de documentos

Exemplo: O exemplo abaixo escreve um elemento `<p>` diretamente na produção de documentos HTML.

```
<!DOCTYPE html>
<html>
<body>

<h1>Minha primeira página web</h1>

<script>
document.write("<p>Meu primeiro Javascript</p>");
</script>

</body>
</html>
```

2.3. Aviso

Use `document.write()` apenas para escrever diretamente na produção de documentos.

Se você executar `document.write` após o documento foi carregado, toda a página HTML será substituída. Exemplo:

```
<!DOCTYPE html>
<html>
<body>

<h1>Minha primeira página</h1>

<p>Meu primeiro parágrafo</p>

<button onclick="myFunction()">Experimente</button>

<script>
function myFunction()
{
  document.write("Oops! O documento desapareceu.");
}
</script>

</body>
</html>
```

2.4. Console

Você pode escrever informações pelo console do Javascript. Os outputs podem ser vistos no painel “Console” do navegador e não aparecerão no documento HTML.

```
<!DOCTYPE html>
<html>
<body>

<script>
console.log("Olá. Olha eu aqui!");
</script>

</body>
</html>
```

JavaScript no Windows

Microsoft suporta JavaScript para a criação de aplicativos desde o Windows 8. JavaScript é definitivamente o futuro, tanto para a Internet e Windows.

3. Funções

Uma função é um bloco de código que será executado quando "alguém" lhe chama. Exemplo:

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction()
{
  alert("Olá mundo!");
}
</script>
</head>

<body>
<button onclick="myFunction()">Experimente</button>
</body>
</html>
```

3.1. Sintaxe da função JavaScript

A função é escrita como um bloco de código (dentro de chaves “{ }”), precedida pelo nome da função e a palavra-chave “`function`”:

```
function nomeDaFuncao()
{
   // some code to be executed
}
```

O código dentro da função será executado quando "alguém" chama a função.

A função pode ser chamada diretamente quando ocorre um evento (como quando um usuário clica em um botão), e pode ser chamado de "qualquer lugar" pelo código JavaScript.

JavaScript é *case-sensitive*. O nome da função deve ser escrito em letras minúsculas, e a função deve ser chamada com os mesmos capitais como usado no nome da função.

3.2. Chamar uma função com argumentos

Quando você chamar uma função, você pode passar ao longo de alguns valores para ele, estes valores são chamados argumentos ou parâmetros. Estes argumentos podem ser usados dentro da função.

Você pode enviar quantos argumentos quiser, separados por vírgulas (,)

```
myFunction(argument1,argument2)
```

Declare o argumento, como variáveis, quando você declarar a função:

```
function myFunction(var1,var2)
{
  // seu código
}
```

As variáveis e os argumentos devem estar na ordem esperada. A primeira variável é o valor do primeiro argumento passado, etc. Exemplo:

```
<button onclick="myFunction('Harry Potter','Wizard')">
Clique aqui </button>

<script>
   function myFunction(name,job)
   {
   alert("Welcome " + name + ", the " + job);
   }
</script>
```

A função acima irá alertar "Welcome Harry Potter, the Wizard" quando o botão é clicado.

A função é flexível: você pode chama-la usando argumentos diferentes, e outras mensagens de boas-vindas serão informadas. Exemplo:

```
<button onclick="myFunction('Harry Potter','Wizard')">Try
it</button>
<button onclick="myFunction('Bob','Builder')">Try it</button>
```

O exemplo acima irá alertar "Welcome Harry Potter, the Wizard" ou "Welcome Bob, the Builder", dependendo de qual botão é clicado.

3.3. Funções com um valor de retorno

Às vezes você quer a sua função para retornar um valor de volta para onde a chamada foi feita. Isso é possível utilizando o comando “return”.

Ao utilizar o retorno, a função irá parar a execução, e retornar o valor especificado.

Sintaxe:

```
function myFunction()
{
  var x=5;
  return x;
}
```

A função acima irá retornar o valor 5.

Nota: Não é toda a JavaScript que irá parar a execução, somente a função. O algoritmo irá continuar a execução do código, onde a função de chamada foi feita.

A função de chamada será substituído com o valor de retorno:

```
var myVar = myFunction();
```

O `myVar` variável tem o valor 5, que é o que a função "`myfunction()`" retorna.

Você também pode usar o valor de retorno, sem armazená-lo como uma variável:

```
document.getElementById("demo").innerHTML = myFunction();
```

O `innerHTML` do elemento "demo" será de 5, que é o que a função "myfunction()" retorna.

Você pode fazer um valor de retorno com base em argumentos passados para a função.

Exemplo: calcular o produto de dois números e retornar o resultado.

```
function myFunction(a,b)
{
  return a*b;
}
document.getElementById("demo").innerHTML=myFunction(4,3);
```

O `innerHTML` do elemento "demo" será:

```
12
```

A declaração de retorno também é usada quando você simplesmente quer sair de uma função. O retorno valor é opcional:

```
function myFunction(a,b)
{
if (a>b)
  {
  return;
  }
x=a+b
}
```

A função acima irá sair da função de se a>b, e não calcula a soma de a e b.

3.4. Variáveis locais

Uma variável declarada (usando var) dentro de uma função JavaScript torna LOCAL e só pode ser acessado de dentro dessa função. (A variável tem escopo local).

Você pode ter variáveis locais com o mesmo nome em diferentes funções, pois as variáveis locais só são reconhecidas pela função em que são declaradas.

As variáveis locais são eliminadas logo que a função é completada.

3.5. Variáveis globais

Variáveis declaradas fora de uma função, se GLOBAL, todos os scripts e funções na página web pode acessá-lo.

3.6. O tempo de vida das variáveis

A vida das variáveis JavaScript começa quando ela é declarada.

As variáveis locais são excluídas quando a função está concluída.

As variáveis globais são excluídas quando você fechar a página.

3.7. Atribuir valores às variáveis de JavaScript não declarados

Se você atribuir um valor a variável que ainda não foi declarada, a variável será automaticamente declarada como uma variável GLOBAL.

```
carname="Volvo";
```

Essa linha declara a variável `carname` como uma variável global, mesmo se for executado dentro de uma função.

3.8. Outra forma de declarar uma função

Você pode encontrar uma função em exemplos com outro formato trabalhado até então. Para quem já sabe como definir uma variável, é muito simples:

```
var funcao1 = function( ) {
   document.write("Olá mundo!");
};

var funcao2 = function(par1) {
   document.write(par1);
};

funcao1();
funcao2("Brackman");
```

Exercícios

1. Crie uma função que calcule a média aritmética de duas notas e mostre o resultado ao apertar um botão.

4. Onde inserir o código Javascript

JavaScripts podem ser colocados no `<body>` e nas seções `<head>` de uma página HTML.

4.1. JavaScript em `<body>`

O exemplo abaixo manipula o conteúdo de um elemento `<p>` existente quando a página é carregada:

```
<html>
<body>
<h1>Minha página</h1>

<p id="demo">Um parágrafo</p>

<script type="text/javascript">
  document.getElementById("demo").innerHTML="Meu teste de
javascript";
</script>

</body>
</html>
```

Nota: O JavaScript é colocado na parte inferior da página para se certificar de que não é executado antes do elemento `<p>` ser criado.

4.2. Funções JavaScript e Eventos

A declaração JavaScript no exemplo acima é executada quando a página é carregada, mas que nem sempre é o que queremos.

Às vezes queremos executar um JavaScript quando ocorre um evento, como por exemplo, quando um usuário clica em um botão.

Para isto é recomendável que você coloque o script dentro de uma **função**.

As funções são normalmente usadas em combinação com **eventos**.

Você vai aprender mais sobre as funções de JavaScript e eventos em mais adiante.

4.3. Funções de JavaScript em `<head>`

O exemplo a seguir chama uma função quando um botão é clicado:

Exemplo

```
<html>
<head>
<script type="text/javascript">
function myFunction() {
   document.write("Função javascript no HEAD");
}
</script>
</head>
```

```
<body>

<button type="button" onclick="myFunction()">Clique aqui</button>

</body>
</html>
```

4.4. Funções de JavaScript em `<body>`

Este exemplo também chama uma função quando um botão é clicado, mas o script é colocado na parte inferior da página:

Exemplo

```
<html>
<body>
<h1>Minha página web</h1>

<p id="demo">Um parágrafo</p>

<button type="button" onclick="myFunction()">Clique aqui</button>

<script type="text/javascript">
function myFunction() {
document.getElementById("demo").innerHTML="Minha função javascript
no BODY";
}
</script>

</body>
</html>
```

4.5. Scripts em `<head>` e `<body>`

Você pode colocar um número ilimitado de scripts em seu documento, e você pode ter scripts no corpo e na seção cabeçalho, ao mesmo tempo. É uma prática comum colocar todas as funções no cabeçalho, ou na parte inferior da página. Dessa forma, eles estão todos em um lugar e não vão interferir com o conteúdo da página.

4.6. Usando um JavaScript externo

JavaScript também pode ser colocado em arquivos externos. Arquivos externos JavaScript frequentemente contêm código para ser usado em várias páginas web diferentes. Arquivos JavaScript externos têm a extensão de arquivo '.js'.

Para usar um arquivo externo, aponte para o arquivo js no atributo "`src`" da *tag* `<script>`. Veja o exemplo a seguir:

```
<html>
<body>
   <script type="text/javascript" src="myScript.js"></script>
</body>
</html>
```

Nota1: Você pode colocar o script no cabeçalho ou no corpo.

Nota2: O script irá se comportar como ele foi localizado no documento (exatamente onde você o inseriu). Exemplo: `document.write()`

Script externo não pode conter as *tags* `<script>` e `</script>`!

Exercícios

1. Crie uma função em um arquivo externo que calcule a média aritmética de três notas e mostre o resultado ao apertar um botão.

5. Operadores JavaScript

O operador de atribuição = é usado para atribuir valores a variáveis de JavaScript.

O operador aritmético + é usado para adicionar valores.

Exemplo de como Atribuir valores às variáveis e adicioná-los juntos:

```
y=5;
z=2;
x=y+z;
```

O resultado de x será: 7

5.1. Operadores aritméticos JavaScript

Os operadores aritméticos são usados para executar aritmética entre as variáveis e/ou valores. Dado que y = 5, a tabela abaixo explica os operadores aritméticos:

Operador	Descrição	Exemplo	Resultado de x	Resultado de y
+	Adição	x = y +2	7	5
-	Subtração	x = y-2	3	5
*	Multiplicação	x = y * 2	10	5
/	Divisão	x = y / 2	2,5	5
%	Módulo (resto da divisão)	x = 2 % y	1	5
++	Incrementa	x = ++y	6	6
		x = y++	5	6
--	Decrementa	x = --y	4	4
		x = y--	5	4

5.2. Operadores de atribuição JavaScript

Os operadores de atribuição são utilizados para atribuir valores à variáveis de JavaScript. Dado que **x=10 e y=5**, a tabela abaixo explica os operadores de atribuição:

Operador	Exemplo	Equivalente	Resultado
=	x = y	-	x=5
+=	x += y	x = x+y	x=15
-=	x -= y	x = x-y	x=5
*=	x *= y	x = x*y	x=50
/=	x /= y	x = x/y	x=2
%=	x %= y	x = x%y	x=0

5.3. O Operador + usado em strings

O operador + pode também ser usado para adicionar variáveis de cadeia ou valores de texto em conjunto.

Para adicionar duas ou mais variáveis de cadeia juntos, use o operador +.

```
txt1 = "What a very";
txt2 = "nice day";
txt3 = txt1 + " " + txt2;
```

O resultado de txt3 será:

```
What a very nice day
```

5.4. Concatenando Strings e Números

Adicionando dois números, irá retornar a soma, mas a adição de um número e uma string irá retornar uma string:

```
x = 5 + 5;
y = "5" + 5;
z = "Hello" + 5;
```

O resultado de x, y e z será:

```
10
55
Hello5
```

A regra é: Se você adicionar um número e uma *string*, o **resultado será uma *string*!**

Exercícios

1. Crie uma função que receba dois números e realize o cálculo de todos os operadores trabalhos na aula;
2.

6. Operadores Lógicos e de Comparação

Os operadores de comparação e lógicos são usados para testar se o resultado é verdadeiro ou falso.

6.1. Operadores de Comparação

Os operadores de comparação são usados em expressões lógicas para determinar a igualdade ou a diferença entre as variáveis ou valores.

Dado que **x = 5**, a tabela abaixo explica os operadores de comparação:

Operador	Descrição	Comparação	Retorno
==	É igual	x == 8	*false*
		X == 5	*true*
===	É exatamente igual (valor e tipo)	x === "5"	*false*
		x === 5	*true*
!=	Diferente	x != 8	*true*
!==	Diferente (valor e tipo)	x !== "5"	*true*
		x !== 5	*false*
>	É maior que	x > 8	*false*
<	É menor que	x < 8	*true*
>=	É maior ou igual que	x >= 8	*false*
<=	É menor ou igual que	x <= 8	*true*

6.2. Como ele pode ser usado

Os operadores de comparação podem ser utilizados em comandos condicionais para comparar valores e tomar medidas, dependendo do resultado:

```
if (age<18) x = "Menor de idade";
```

Você vai aprender mais sobre o uso de declarações condicionais mais adiante.

6.3. Operadores lógicos

Os operadores lógicos são utilizados para determinar a lógica entre as variáveis ou valores. Dado que **x = 6 e y = 3**, a tabela abaixo explica os operadores lógicos:

Operador	Descrição	Exemplo
&&	and	(x < 10 && y > 1) is true
||	or	(x==5 || y==5) is false
!	not	!(x==y) is true

6.4. Operador condicional

JavaScript também contém um operador condicional que atribui um valor a uma variável baseada em alguma condição.

Sintaxe

```
nome_variavel = (condicao)? valor1:valor2
```

Exemplo

Se a variável ‘age’ é um valor abaixo de 18, o valor da variável ‘pode_dirigir’ será "muito jovem”, caso contrário o valor de ‘pode_dirigir’ será "idade suficiente":

```
Voteable = (age<18)?"muito jovem":"idade suficiente";
```

7. If ... Else

Declarações condicionais são usadas para executar ações diferentes baseadas em diferentes condições.

7.1. Declarações Condicionais

Muitas vezes, quando você escrever o código, você pode querer executar diferentes ações para diferentes decisões. Você pode usar instruções condicionais no código para fazer isso.

Em JavaScript temos as seguintes declarações condicionais:

- **if** - use esta instrução para executar algum código somente se uma condição especificada é verdadeira

```
if (condição)
{
  // código a ser executado caso a condição for VERDADEIRA
}
```

- **if...else** - use esta instrução para executar algum código se a condição for verdadeira e outro código se a condição for falsa

```
if (condição)
{
  // código a ser executado caso a condição for VERDADEIRA
}
else
{
  // código a ser executado caso a condição for FALSA

}
```

- **if...else if....else** - use esta instrução para selecionar um dos muitos blocos de código a ser executado

```
if (condição1)
{
  // código a ser executado caso a condição1 for VERDADEIRA
}
else if (condição2)
{
  // código a ser executado caso a condição2 for VERDADEIRA
}
else
{
     // código a ser executado caso a condição1 e condição2
     // forem ambas VERDADEIRAS
}
```

8. Switch

A instrução switch é usada para executar diferentes ações com base em diferentes condições.

8.1. A instrução switch JavaScript

Use a instrução switch para selecionar um dos muitos blocos de código a ser executado.

O switch pode ser substituído por muitas condições “if-else”, mas quando a resposta de um teste pode lhe dar muitas respostas diferentes, o “switch-case” pode ser mais eficiente.

Sintaxe

```
switch(n)
{
case 1:
  executa bloco de execução 1
  break;
case 2:
  executa bloco de execução 2
  break;
default:
  bloco a ser executado se n é diferente de 1 e 2
}
```

É assim que funciona: Primeiro, temos uma única expressão n (na maioria das vezes uma variável), que é avaliada uma vez. O valor da expressão é então comparada com os valores para cada caso na estrutura (“case”). Se houver uma correspondência, o bloco de código associado com esse caso é executado. Use “break” para impedir que o código seja executado no próximo caso automaticamente.

Exemplo

Mostre o nome do dia da semana de hoje. Note que domingo = 0, segunda-feira = 1, terça-feira = 2, etc.:

```
var day=new Date().getDay();
switch (day)
{
case 0:
  x="Hoje é domingo";
  break;
case 1:
  x=" Hoje é segunda-feira";
  break;
case 2:
  x=" Hoje é terça-feira";
  break;
case 3:
```

```
  x=" Hoje é quarta-feira";
  break;
case 4:
  x=" Hoje é quinta-feira";
  break;
case 5:
  x=" Hoje é sexta-feira";
  break;
case 6:
  x=" Hoje é sábado";
  break;
}
document.write(x);
```

O resultado de x, por exemplo, será:

```
Hoje é sexta-feira
```

8.2. O caso “default”

Use a palavra-chave “default” para especificar o que fazer, se não houver correspondência em nenhum dos casos anteriores.

Exemplo:

Se não é nem Sábado ou Domingo, escreva uma mensagem.

```
var day = new Date().getDay();
switch (day)
{
case 6:
  x="Hoje é Sábado";
  break;
case 0:
  x="Hoje é Domingo";
  break;
default:
  x="Estou ansioso para a chegada do final de semana ;-)";
}
document.write(x);
```

O resultado de x será:

```
Estou ansioso para a chegada do final de semana ;-)
```

9. Laços

Os laços vão executar um bloco de código um número especificado de vezes.

9.1. Laços em Javascript

Muitas vezes, você precisa que o mesmo bloco de código seja executado várias vezes na sequência. Em vez de adicionar várias linhas parecidas em um script, podemos usar laços para executar uma tarefa como esta.

Em JavaScript, existem dois diferentes tipos de loops:

- **for** – executa o bloco de código um determinado número de vezes (pré-determinado);
- **while** - percorre um bloco de código enquanto uma condição especificada é verdadeira (quantidade de vezes indeterminada). O teste pode ser feito no início ou no final do bloco.

9.2. O laço FOR

O loop for é usado quando você sabe de antemão quantas vezes o script deve ser executado.

Sintaxe

```
for (variavel = valor_inicial; variavel < valor_final; variavel =
variavel+incremento)
{
  // bloco de código a ser executado
}
```

Exemplo 1

O exemplo abaixo define um laço que começa com i=0. O ciclo continuará executando enquanto i for menor que 5. i vai incrementar em 1 cada vez que o laço é executado.

Nota: O parâmetro incremento poderia também ser negativo, e o < poderia ser qualquer instrução de comparação.

```
for (i=0; i<5; i++)
{
  x=x + "The number is " + i + "<br />";
}
```

Exemplo 2

O código a seguir deverá mostrar os 6 níveis de seções da sua página.

```
<html>
<body>
<button onclick="myFunction()">Clique aqui</button>
<div id="demo"></div>
```

```
<script type="text/javascript">
function myFunction()
{
var x="",i;
for (i=1; i<=6; i++)
{
x += "<h" + i + ">Seção " + i + "</h" + i + ">";
}
document.getElementById("demo").innerHTML=x;
}
</script>
</body>
</html>
```

9.3. O laço WHILE (verifica condição no início do bloco)

O laço while repete um bloco de código enquanto uma condição especificada for verdadeira.

Sintaxe

```
while (variavel < valor_final)
{
  // código a ser executado
}
```

Nota: O < poderia ser qualquer operador de comparação.

Exemplo

O exemplo abaixo define um laço que começa com i = 0. O ciclo continuará executando enquanto i for menor que 5. i vai incrementar em 1 cada vez que o loop é executado:

```
while (i < 5)
{
  x=x + "O número é " + i + "<br />";
  i++;
}
```

Nota: Não se esqueça de aumentar o i variável utilizada na condição de, caso contrário, i sempre vai ser inferior a 5, eo ciclo nunca vai acabar!

9.4. O laço DO ... WHILE (verifica a condição no final do bloco)

O laço do...while é uma variante do laço while. Este laço será executa o bloco de código uma vez, antes de verificar se a condição for verdadeira, então ele vai repetir o loop enquanto a condição for verdadeira.

Sintaxe

```
do
{
  // código a ser executado
}
while (variavel < valor_final);
```

Exemplo

O exemplo abaixo usa um do...while. O laço do..while será sempre executado pelo menos uma vez, mesmo se a condição for falsa, porque as instruções são executadas antes da condição ser testada:

```
do
{
  x=x + "O número é " + i + "<br />";
  i++;
}
while (i<5);
```

Nota: Não se esqueça de incrementar da variável i, pois esta variável é utilizada na condição. Caso você não incremente a variável, i sempre vai ser inferior a 5, e o ciclo nunca vai acabar (é o famoso “looping infinito”).

9.5. Comparando for e while

O laço neste exemplo usa um loop while para exibir todos os valores na matriz carros, ou seja, não há necessidade de saber o tamanho do array. Exemplo:

```
cars=["BMW","Volvo","Saab","Ford"];
var i=0;
while (cars[i])
{
document.write(cars[i] + "<br>");
i++;
}
```

10. Primeiros Programas

JavaScript é uma sequência de instruções a serem executadas pelo browser.

10.1. Demonstrações JavaScript

Uma declaração de JavaScript é um comando para um navegador. O objetivo do comando é para informar ao navegador o que fazer.

Esta declaração informa ao navegador JavaScript para escrever "Hello World" a um elemento HTML:

```
document.getElementById("demo").innerHTML="Hello World!";
```

Ponto e Vírgula

É normal adicionar um ponto e vírgula no final de cada instrução executável. A maioria das pessoas pensa que esta é apenas uma boa prática de programação, e na maioria das vezes você vai ver isso em exemplos de JavaScript na web.

A vírgula é opcional (de acordo com a norma JavaScript), e é suposto pelo navegador de interpretar o fim da linha como o fim da instrução. Devido a isso muitas vezes você vai ver exemplos sem a vírgula no final.

Nota: Usando ponto e vírgula torna possível escrever várias instruções em uma linha.

10.2. Código JavaScript

O código JavaScript (ou apenas JavaScript) é uma sequência de instruções JavaScript.

Cada instrução é executada pelo navegador na sequência em que são escritos.

Este exemplo irá manipular dois elementos HTML:

```
document.getElementById("demo").innerHTML="Hello Dolly";
document.getElementById("myDIV").innerHTML="How are you?";
```

10.3. Blocos de JavaScript

Declarações JavaScript podem ser agrupadas em blocos.

Blocos começam com uma chave na esquerda ({), e terminam com uma chave na direita (}). A finalidade de um bloco é fazer com que a sequência de instruções sejam executadas em conjunto.

Um exemplo de declarações agrupadas em blocos, são JavaScript **funções**.

Este exemplo irá executar uma função que irá manipular elementos HTML:

```
function myFunction()
{
document.getElementById("demo").innerHTML="Hello Dolly";
document.getElementById("myDIV").innerHTML="How are you?";
}
```

Você vai aprender mais sobre as funções e as condições em mais adiante.

10.4. JavaScript é sensível a maiúsculas

JavaScript é case sensitive.

Cuidado com a capitalização de perto quando você escreve instruções JavaScript:

- Um getElementById função não é o mesmo que GetElementById.
- Uma variável minhaVariavel chamada não é o mesmo que MinhaVariavel.

10.5. Espaço em Branco

JavaScript ignora espaços extras. Você pode adicionar um espaço em branco para o seu script para torná-lo mais legível. As linhas a seguir são equivalentes:

```
var person="Hege";
var person = "Hege";
```

10.6. Quebrar uma linha de código

Você pode acabar com uma linha de código dentro de uma cadeia de texto com uma barra invertida. O exemplo abaixo será exibido corretamente:

```
document.write("Hello \
World!");
```

No entanto, você não pode acabar com uma linha de código como este:

```
document.write \
("Hello World!");
```

11. Comentários

Os comentários não serão executados pelo JavaScript.

Comentários podem ser adicionados para explicar o JavaScript, ou para tornar o código mais legível.

Comentários de linha única começa com //.

O exemplo a seguir usa comentários de linha única para explicar o código:

```
// Escreve no cabeçalho:
document.getElementById("myH1").innerHTML = "Bem Vindo!";
// Escreve em um parágrafo:
document.getElementById("myP").innerHTML="Meu parágrafo";
```

11.1. Comentário Multilinha

Comentários de mais de uma linha começa com / * e terminam com * /.

O exemplo a seguir usa um comentário multilinha para explicar o código:

```
/*
Escreve no cabeçalho
e logo após o parágrafo.
*/

document.getElementById("myH1").innerHTML = "Bem Vindo!";
document.getElementById("myP").innerHTML="Meu parágrafo";
```

11.2. Usando comentários para impedir a execução

No exemplo seguinte, o comentário é usado para impedir a execução de um dos codelines (pode ser adequado para a depuração):

```
//document.getElementById("myH1").innerHTML = "Bem Vindo!";
document.getElementById("myP").innerHTML="Meu parágrafo";
```

No exemplo seguinte, o comentário é usado para impedir a execução de um bloco de código (pode ser adequado para a depuração):

```
/*
document.getElementById("myH1").innerHTML = "Bem Vindo!";
document.getElementById("myP").innerHTML="Meu parágrafo";
*/
```

11.3. Usando comentários no final de uma linha

No exemplo seguinte, o comentário é colocado na extremidade de uma linha de código:

```
var x=5; // declara uma variável e atribui um valor a ela
x=x+2;   // adiciona o número 2 a variavel
```

12. Variáveis

As variáveis são containers para armazenar a informação.

12.1. Você se lembra de Álgebra?

x = 5,

y = 6,

z = x + y

Em álgebra usamos letras (como x) para manter os valores (como o 5).

Pela expressão z = x + y acima, pode-se calcular o valor de z que é 11.

Em JavaScript essas letras são chamadas de variáveis.

12.2. Variáveis em JavaScript

Tal como acontece com a álgebra, as variáveis de JavaScript são usadas para armazenar valores ou expressões.

Uma variável pode ter um nome curto, como x, ou um nome mais descritivo, como nome_completo.

Regras para nomes de variáveis JavaScript:

- Nomes de variáveis devem começar com uma letra
- Os nomes das variáveis também pode começar com $ e _ (mas não vamos usá-los)
- Os nomes das variáveis são case-sensitive (y e Y são variáveis diferentes)

12.3. Tipos de dados JavaScript (introdução)

Variáveis JavaScript também pode conter outros tipos de dados, como valores de texto (pessoa = "John Doe").

Em JavaScript um texto como "John Doe" é chamado de string.

Existem muitos tipos de variáveis de JavaScript, mas por enquanto, só pensa em números e strings.

Quando você atribui um valor de texto a uma variável, coloque aspas duplas ou simples em torno do valor.

Quando você atribui um valor numérico a uma variável, não coloque aspas em torno do valor. Se você colocar aspas em torno de um valor numérico, ele será tratado como texto.

Exemplo:

```
var pi=3.14;
var person="John Doe";
var answer='Yes I am!';
```

No capítulo seguinte iremos trabalhar todos os tipos de dados que o JavaScript trabalha.

12.4. Declarando (Criação) Variáveis em JavaScript

Criar uma variável em JavaScript é frequentemente referido como "declarar uma variável". Você declara variáveis em JavaScript com o "`var nome_da_variavel`":

```
var carname;
```

Após a declaração acima, a variável está vazia (não tem valor ainda).

Para atribuir um valor à variável, use o sinal de igual (=):

```
carname = "Volvo";
```

No entanto, você também pode atribuir um valor à variável ao declará-la:

```
var carname = "Volvo";
```

Após a execução da declaração acima, o "`carname`" vai conter o valor "Volvo".

Para escrever o valor dentro de um elemento HTML, simplesmente se refira a ele usando o nome da variável:

```
<p id="demo"></p>
var carname="Volvo";
 document.getElementById("demo").innerHTML=carname;
```

Nota: Quando você atribui um valor de texto (*string*) a uma variável, coloque aspas em torno do valor.

Nota: Quando você atribui um valor numérico a uma variável, não coloque aspas em torno do valor, se você colocar aspas em torno de um valor numérico, ele será tratado como texto.

É uma boa prática de programação declarar todas as variáveis que você precisa, em um só lugar, no início do seu código.

12.5. Variáveis locais JavaScript

Uma variável declarada dentro de uma função JavaScript torna-se LOCAL e só pode ser acessada dentro daquela função (a variável tem escopo local).

Você pode ter variáveis locais com o mesmo nome em funções diferentes, porque as variáveis locais só são reconhecidas pela função em que elas são declaradas.

As variáveis locais são suprimidas, logo que a função é concluída.

12.6. Variáveis globais JavaScript

Variáveis declaradas fora de uma função se tornam GLOBAIS, ou seja, todos os scripts e as funções na página web podem acessá-la.

As variáveis globais são apagadas quando você fecha a página.

12.7. Atribuindo valores a variáveis não declaradas JavaScript

Se você atribuir valores a variáveis que ainda não tenham sido declaradas, as variáveis serão automaticamente declaradas como variáveis globais.

Esta declaração vai declarar a variável carname como uma variável global (se já não existir):

```
carname="Volvo";
```

12.8. Uma declaração, muitas variáveis

Você pode declarar diversas variáveis em um comunicado. Basta começar a declaração com var e separar as variáveis por vírgula:

```
var lastname="Doe", age=30, job="carpenter";
```

Sua declaração também pode abranger várias linhas:

```
var lastname="Doe",
age=30,
job="carpenter";
```

12.9. Valor = indefinido

Em programas de computador, as variáveis são muitas vezes declarada sem um valor. O valor pode ser algo que tem de ser calculado, ou algo que será fornecido mais tarde, como a entrada do usuário. Variável declarada sem um valor terá o valor indefinido.

A variável carname terá o valor indefinido após a execução da seguinte declaração:

```
var carname;
```

12.10. Re-Declarando Variáveis JavaScript

Se você voltar a declarar uma variável JavaScript, ele não vai perder o seu valor:.

O valor da variável carname ainda terá o valor " Volvo ", após a execução das duas instruções a seguir:

```
var carname="Volvo";
var carname;
```

12.11. Aritmética JavaScript

Tal como acontece com a álgebra, você pode fazer operações aritméticas com variáveis JavaScript:

```
y=5;
x=y+2;
```

Você vai aprender mais sobre os operadores que podem ser utilizados no próximo capítulo.

12.12. Acessando o conteúdo de INPUTS

Você pode acessar os elementos de INPUTS pela linha de código:

```
var_ret = document.getElementById("[id do objeto]").value;
```

Note que pode ser necessário tratar os retornos, pois eles poderão ser do tipo string. Caso precise extrair (converter) um número de string para inteiro, utilize a seguinte método:

```
var_ret = parseInt( document.getElementById("[id do objeto]").value
);
```

13. Tipos de Dados

String, Number, Boolean, Array, Object, Null, Undefined.

13.1. JavaScript Tem tipos dinâmicos

JavaScript tem tipos dinâmicos. Isto significa que a mesma variável pode ser usada como diferentes tipos. Exemplo:

```
var x;               // Now x is undefined
var x = 5;           // Now x is a Number
var x = "John";      // Now x is a String
```

13.2. String

Uma string é uma variável que armazena uma série de personagens, como "John Doe". A sequência pode ser qualquer texto entre aspas. Você pode usar aspas simples ou duplas. Exemplo:

```
var carname="Volvo XC60";
var carname='Volvo XC60';
```

Você pode usar aspas dentro de uma string, contanto que não coincidem com as aspas da string. Exemplo:

```
var answer="It's alright";
var answer="He is called 'Johnny'";
var answer='He is called "Johnny"';
```

13.3. Números

JavaScript possui apenas um tipo de números. Os números podem ser escritos com ou sem casas decimais. Exemplo:

```
var x1=34.00;      // com decimal
var x2=34;         // sem decimal
```

Números extremamente grandes ou pequenos podem ser escritos com a notação científica (exponencial). Exemplo:

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

13.4. Booleanos

Booleanos só pode ter dois valores: verdadeiro ou falso.

```
var x=true;
var y=false;
```

Booleanos são frequentemente usados em teste condicional.

13.5. Arrays

O código a seguir cria uma matriz chamada carros:

```
var cars=new Array();
cars[0]="Saab";
cars[1]="Volvo";
cars[2]="BMW";
```

Ou (array condensado):

```
var cars=new Array("Saab","Volvo","BMW");
```

Ou (array literal):

```
var cars=["Saab","Volvo","BMW"];
```

Índices de matriz são baseados em zero, o que significa que o primeiro item é [0], o segundo é [1], e assim por diante.

13.6. Objetos

Um objeto é delimitado por chaves. Dentro das chaves as propriedades do objeto são definidos como pares de nome e valor (nome: valor). As propriedades estão separados por vírgulas:

```
var person={firstname:"John", lastname:"Doe", id:5566};
```

O objeto (pessoa) no exemplo acima tem três atributos: nome, sobrenome, e ID.

Espaços e quebras de linha não são importantes. Sua declaração pode abranger várias linhas:

```
var person={
firstname : "John",
lastname  : "Doe",
id        :  5566
};
```

Você pode acessar as propriedades do objeto de duas maneiras. Exemplo:

```
name=person.lastname;
name=person["lastname"];
```

13.7. Indefinido e Null

Indeterminado é o valor de uma variável sem valor.

As variáveis podem ser esvaziado, definindo o valor para nulo. Exemplo:

```
cars=null;
person=null;
```

13.8. Declarando Tipos de variáveis

Quando você declarar uma nova variável, você pode declarar seu tipo usando a palavra-chave "new":

```
var carname = new String;
var x =       new Number;
var y =       new Boolean;
var cars =    new Array;
var person =  new Object;
```

Variáveis JavaScript são todos objetos. Quando você declara uma variável, você cria um novo objeto.

14. Objetos em JavaScript

“Tudo” em JavaScript é um objeto: uma string, um número, uma matriz, uma data...

Em JavaScript, um objeto é composto de dados, propriedades e métodos.

14.1. Propriedades e métodos

Propriedades são valores associados a um objeto.

Métodos são ações que podem ser executadas em objetos.

14.2. Um objeto da Vida Real: um carro

Propriedades:	**Métodos:**

`car.name = Fiat`
`car.model = 500`
`car.weight = 850kg`
`car.color = branco`
	`car.start()`
	`car.drive()`
	`car.brake()`

As propriedades do carro incluir o nome, o modelo, o peso, a cor, etc

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

Os métodos do carro poderiam ser start(), drive(), break(), etc.

Todos os carros possuem esses métodos, mas eles são realizados em momentos diferentes.

14.3. Objetos em JavaScript:

Em JavaScript, os objetos são dados (variáveis), com propriedades e métodos.

Quando você declara uma variável JavaScript como esta:

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

Na verdade você cria um objeto string. O objeto string possui um atributo adicional chamado length (comprimento). Para a sequência acima, o comprimento tem o valor 5. O objeto string também possui vários métodos internos.

Propriedades:	**Métodos:**

`txt.length = 5`	`txt.indexOf()`
	`txt.replace()`
	`txt.search()`

Em linguagens orientadas a objetos, propriedades e métodos são frequentemente chamados de membros do objeto.

14.4. Criação de objetos JavaScript

Quase "tudo" em JavaScript é um objeto. Strings, datas, matrizes, funções. Você também pode criar seus próprios objetos. Este exemplo cria um objeto chamado "pessoa", e acrescenta quatro atributos a ele:

```
person=new Object();
person.firstname="John";
person.lastname="Doe";
person.age=50;
person.eyecolor="blue";
```

Há muitas maneiras diferentes para criar novos objetos, e você também pode adicionar atributos e métodos de objetos existentes.

14.5. Acessando propriedades de objetos

A sintaxe para acessar a propriedade de um objeto é:

```
objectName.propertyName
```

Este exemplo usa a propriedade length do objeto String para encontrar o comprimento de uma string:

```
var message="Hello World!";
var x=message.length;
```

O valor de x, depois da execução do código acima será: 12

14.6. Acessando Métodos de Objeto

Você pode chamar um método com a seguinte sintaxe:

```
objectName.methodName()
```

Este exemplo usa o método do objeto String toUpperCase (), para converter um texto em maiúsculas:

```
var message="Hello world!";
var x=message.toUpperCase();
```

O valor de x, depois da execução do código acima será: `HELLO WORLD!`

Você Sabia?

É comum em linguagens orientadas a objeto, para dar nome às funções se usa “camel-case”.

Você vai ver com mais frequência nomes de funções como someMethod() em vez de some_method().

15. Caixas de Diálogo

JavaScript tem três tipos de caixas de diálogo: Caixa de Alerta, Caixa de Confirmação e Caixa de Prompt(pergunta).

15.1. Caixa de Alerta

Uma caixa de alerta é frequentemente usada para você ter certeza que o usuário teve conhecimento de alguma informação.

Quando uma caixa de alerta aparece, o usuário terá de clicar em "OK" para prosseguir.

Sintaxe

```
alert("sometext");
```

Exemplo

```
<html>
<head>
<script type="text/javascript">
function myFunction()
{
  alert("Eu sou uma caixa de alerta!");
}
</script>
</head>
<body>

<input type="button" onclick="myFunction()" value="Mostrar a Caixa
de Alerta" />

</body>
</html>
```

15.2. Caixa de Confirmação

Uma Janela de Confirmação é frequentemente usada se você deseja que o usuário verifique ou aceite algo.

Quando a caixa de confirmação aparece, o usuário terá de clicar em "OK" ou "Cancelar" para continuar. Se o usuário clica em "OK", a caixa retorna verdadeiro. Se o usuário clica em "Cancelar", a caixa retorna false.

Sintaxe

```
confirm("algum texto");
```

Exemplo

```
var r = confirm("Pressione um botão");

if (r)
{
```

```
  x = "Você pressionou OK!";
}
else
{
  x = "Você pressionou CANCELAR!";
}
alert(x);
```

15.3. Caixa de Prompt (Pergunta)

A caixa de alerta é muitas vezes usada se você quiser que o usuário insira um valor antes de entrar em uma página.

Quando uma caixa de aviso aparece na tela, o usuário terá de clicar em "OK" ou "Cancelar" para continuar depois de introduzir um valor de entrada.

Se o usuário clica em "OK" a caixa retorna o valor de entrada. Se o usuário clica em "Cancelar" a caixa retorna nulo.

Sintaxe

```
prompt("Algum texto","Resposta-padrão");
```

Exemplo

```
var name = prompt("Por favor, informe seu nome","Shrek");
if (name!=null && name!="")
{
  alert("Olá " + name + "! Como você está?");
}
else
{
  alert("Desculpe, mas não sei seu nome...");
}
```

15.4. Quebras de linha

Para exibir quebras de linha dentro de uma caixa, use uma barra invertida seguida do caractere n.

Exemplo

```
alert("Ola.\nBem-vindo!");
```

Atividades

1) Faça uma função que pergunte ao usuário sua idade. Logo após, diga a qual classe a pessoa pertence:

   0-24 – Classe 1

   25-49 – Classe 2

   50-74 – Classe 3

   Acima de 74 – Classe 4

2) Pergunte ao usuário se ele tem certeza em apagar um usuário. Caso ele aceitar, mostre uma caixa com a mensagem “Você apagou o usuário com sucesso!”. Caso ele tenha desistido, mostre uma mensagem alertando o usuário de que a operação não foi realizada.

16. Break e Continue

Em resumo: a declaração break "salta" de um loop e a instrução continue "pula" uma iteração do loop.

16.1. A instrução break

A instrução break pode ser usada para saltar de um loop.

A instrução break interrompe o ciclo e continua executando o código após o loop (se houver). Exemplo:

```
for (i=0;i<10;i++)
  {
  if (i==3)
    {
    break;
    }
  x=x + "The number is " + i + "<br>";
  }
```

Uma vez que a declaração se tem apenas uma única linha de código, o aparelho pode ser omitido:

```
for (i=0;i<10;i++)
  {
  if (i==3) break;
  x=x + "The number is " + i + "<br>";
}
```

16.2. A instrução continue

A instrução continue interrompe uma iteração (no circuito), se uma condição especificada ocorre, e continua com a próxima iteração do loop.

Este exemplo ignora o valor de 3:

```
for (i=0;i<=10;i++)
 {
 if (i==3) continue;
  x=x + "The number is " + i + "<br>";
  }
```

17. Tratando Erros: Try, Catch e Throw

A declaração try permite que você teste um bloco de código para erros.

A declaração catch permite lidar com o erro.

A declaração throw permite criar erros personalizados.

17.1. Erros vão acontecer!

Quando o engine JavaScript está executando o código JavaScript, diversos erros podem ocorrer:

- Pode haver erros de sintaxe, erros ou enganos feitos pelo programador normalmente codificação.
- Ele pode ser incorreto ou características na língua (talvez devido às diferenças do navegador) faltando.
- Pode haver erros devido à entrada errada, de um usuário, ou a partir de um servidor de Internet.
- E, evidentemente, pode haver muitas outras coisas imprevisíveis.

17.2. JavaScript Lança Erros

Quando ocorre um erro, quando algo dá errado, o engine JavaScript normalmente para, e gerar uma mensagem de erro. O termo técnico para isso é: JavaScript irá lançar um erro.

17.3. JavaScript Tentar Pegar os Erros

A tentativa declaração permite definir um bloco de código a ser testado por erros enquanto ele está sendo executado. A declaração de captura permite definir um bloco de código a ser executado, se ocorrer um erro no bloco try.

As instruções JavaScript tentar e captura vêm em pares.

Sintaxe

```
try
  {
  //Run some code here
  }
catch(err)
  {
  //Handle errors here
  }
```

No exemplo abaixo temos deliberadamente um erro de digitação no código no bloco try.

O bloco catch captura o erro no bloco try, e executa o código para lidar com isso:

```
<!DOCTYPE html>
<html>
<head>
<script>
var txt="";
function message()
{
try
  {
  adddlert("Welcome guest!");
  }
catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
  }
}
</script>
</head>

<body>
<input type="button" value="View message" onclick="message()">
</body>

</html>
```

17.4. A instrução throw

A instrução throw permite criar um erro personalizado. O termo técnico correto é criar uma exceção. Se você usar a instrução throw juntamente com try e catch, você pode controlar o fluxo do programa e gerar mensagens de erro personalizadas.

Sintaxe

```
throw exception
```

A exceção pode ser uma string JavaScript, um número, um valor booleano ou um objeto.

Este exemplo examina o valor de uma variável de entrada. Se o valor está errado, uma exceção (erro) é lançada. O erro é pego pela instrução catch e uma mensagem de erro personalizada é exibida.

```
<script>
function myFunction()
{
try
  {
  var x=document.getElementById("demo").value;
  if(x=="")    throw "empty";
  if(isNaN(x)) throw "not a number";
  if(x>10)     throw "too high";
  if(x<5)      throw "too low";
```

```
  }
catch(err)
  {
  var y=document.getElementById("mess");
  y.innerHTML="Error: " + err + ".";
  }
}
</script>

<h1>My First JavaScript</h1>
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="mess"></p>
```

Note que o exemplo acima também irá lançar um erro se a função getElementById falhar.

18. Validação de Formulário

JavaScript pode ser usado para validar dados em formulários HTML antes de enviar o conteúdo para um servidor.

Os dados do formulário que normalmente são verificados por um JavaScript pode ser:

- se o usuário deixou campos obrigatórios vazios?
- se o usuário digitou um endereço de e-mail é válido?
- se o usuário digitou uma data válida?
- se o usuário digitou texto em um campo numérico?

18.1. Campos obrigatórios

A função abaixo verifica se um campo foi deixado vazio. Se o campo estiver em branco, uma caixa de alerta avisa a mensagem, a função retorna false e o formulário não será submetido:

```
function validateForm()
{
var x=document.getElementById['nome'].value;
 if (x==null || x=="")
 {
   alert("First name must be filled out");
   return false;
 }
}
```

A função acima pode ser chamado quando um formulário é enviado:

```
<form name="myForm" action="demo_form.asp" onsubmit="return
validateForm()" method="post">
First name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

18.2. E-mail de validação

A função abaixo verifica se o conteúdo tem a sintaxe geral de um e-mail.

Isto significa que os dados de entrada devem conter um sinal @ e pelo menos um ponto (.). Além disso, o @ não deve ser o primeiro caractere do endereço de e-mail, eo último ponto deve estar presente após o sinal @, e no mínimo 2 caracteres antes do final:

```
function validateForm()
{
var x=document.forms["myForm"]["email"].value;
var atpos=x.indexOf("@");
var dotpos=x.lastIndexOf(".");
if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)
  {
  alert("Not a valid e-mail address");
  return false;
```

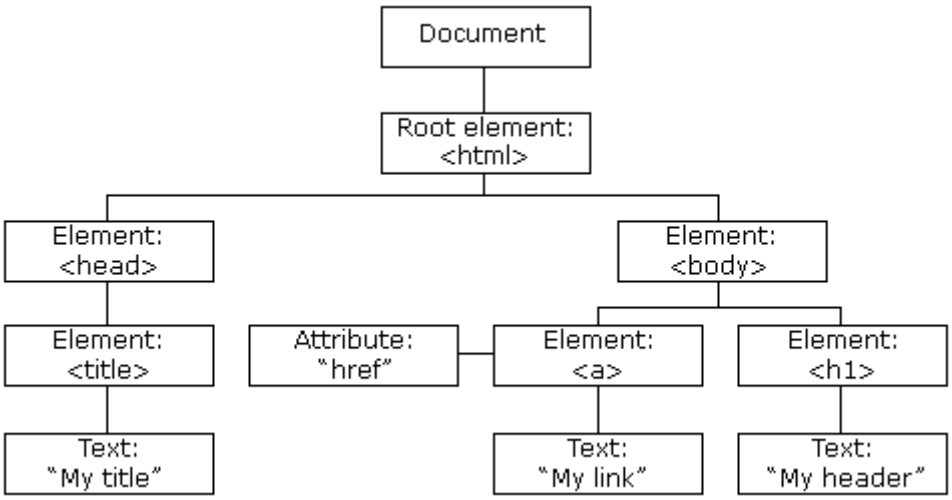
```
  }
}
```

A função acima pode ser chamado quando um formulário é enviado:

```
<form name="myForm" action="demo_form.asp" onsubmit="return
validateForm();" method="post">
Email: <input type="text" name="email">
<input type="submit" value="Submit">
</form>
```

19. DOM (Document Object Model)

Com o DOM HTML, o JavaScript pode acessar todos os elementos de um documento HTML. Quando uma página é carregada, o navegador cria um Document Object Model da página. O modelo DOM HTML é construído como uma árvore de objetos:



Com um modelo de objeto programável, JavaScript recebe todas as ferramentas de que necessita para criar HTML dinâmico:

- JavaScript pode mudar todos os elementos HTML na página
- JavaScript pode mudar todos os atributos HTML na página
- JavaScript pode mudar todos os estilos CSS na página
- JavaScript pode reagir a todos os eventos na página

19.1. Encontrar elementos HTML

Muitas vezes, com JavaScript, você quer manipular elementos HTML. Para fazer isso, você deve em primeiro lugar encontrar os elementos. Existem algumas maneiras de fazer isso:

- Encontrar elementos HTML pelo id
- Encontrar elementos HTML pelo nome da tag
- Encontrar elementos HTML pelo nome da classe

19.2. Encontrar elementos HTML por Id

A maneira mais fácil de encontrar elementos HTML no DOM, é usando o id do elemento. Este exemplo encontra o elemento com id = "intro":

```
var x = document.getElementById("intro");
```

Se o elemento é encontrado, o método irá retornar o elemento como um objeto (em x).

Se o elemento não é encontrado, x conterá nulo.

19.3. Encontrar elementos HTML por Tag

Este exemplo encontra o elemento com id = "main", e em seguida, encontra todos os elementos <p> dentro de "main":

```
var x=document.getElementById("main");
var y=x.getElementsByTagName("p");
```

Encontrar elementos pelo nome da classe não funciona no Internet Explorer 5, 6, 7 e 8.

20. DOM: Mudando o HTML

O HTML DOM permite JavaScript para alterar o conteúdo de elementos HTML.

20.1. Alterando o fluxo de saída HTML

JavaScript pode criar conteúdo HTML dinâmico. Em JavaScript, document.write() pode ser usado para escrever diretamente no fluxo de saída HTML. Exemplo:

```
<!DOCTYPE html>
<html>
<body>

<script>
document.write(Date());
</script>

</body>
</html>
```

Nunca use document.write () depois que o documento é carregado. Ele vai substituir o documento.

20.2. Alterar conteúdo HTML

A maneira mais fácil de modificar o conteúdo de um elemento HTML é usando o atributo innerHTML. Para alterar o conteúdo de um elemento HTML, use a seguinte sintaxe:

```
document.getElementById(id).innerHTML = new HTML
```

Este exemplo modifica o conteúdo de um elemento <p>:

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML="New text!";
</script>

</body>
</html>
Este exemplo altera o conteúdo de um elemento <h1>:
<!DOCTYPE html>
<html>
<body>

<h1 id="header">Old Header</h1>

<script>
var element=document.getElementById("header");
element.innerHTML="New Header";
</script>
```

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

Exemplo explicado:

- O documento HTML acima contém um elemento <h1> com id = "header"
- Usamos o DOM HTML para obter o elemento com id = "header"
- O JavaScript altera o conteúdo (innerHTML) do elemento

20.3. Mudando um atributo HTML

Para alterar o atributo de um elemento HTML, use a seguinte sintaxe:

```
document.getElementById(id).attribute=new value
```

Este exemplo altera o atributo src de um elemento <img>:

```
<!DOCTYPE html>
<html>
<body>

<img id="image" src="smiley.gif">

<script>
document.getElementById("image").src="landscape.jpg";
</script>

</body>
</html>
```

Exemplo explicado:

- O documento HTML acima contém um elemento <img> com id = "imagem"
- Nós usamos o HTML DOM para obter o elemento com id = "imagem"
- O JavaScript muda o atributo src do elemento de "smiley.gif" para "landscape.jpg"

21. DOM: Mudando Propriedades CSS

O HTML DOM permite JavaScript para alterar o estilo de elementos HTML.

21.1. Mudar o estilo HTML

Para alterar o estilo de um elemento HTML, use a seguinte sintaxe:

```
document.getElementById(id).style.property=new style
```

O exemplo a seguir altera o estilo de um elemento <p>:

```
<html>
<body>
<p id="p2">Hello World!</p>
<script>
document.getElementById("p2").style.color="blue";
</script>
<p>The paragraph above was changed by a script.</p>
</body>
</html>
```

Este exemplo altera o estilo do elemento HTML com o id = "id1", quando o usuário clica em um botão:

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id1">My Heading 1</h1>
<button type="button"
onclick="document.getElementById('id1').style.color='red'">
Click Me!</button>

</body>
</html>
```

Este exemplo mostra e esconde um texto, quando o usuário clica em um botão:

```
<!DOCTYPE html>
<html>
<body>
<p id="p1">This is a text. This is a text. This is a text. This is
a text. This is a text. This is a text. This is a text.</p>

<input type="button" value="Hide text"
onclick="document.getElementById('p1').style.visibility='hidden'"
/>
<input type="button" value="Show text"
onclick="document.getElementById('p1').style.visibility='visible'"
/>

</body>
</html>
```

22. DOM: Eventos

HTML DOM permite que o JavaScript para reagir a eventos do HTML.

22.1. Reagir aos acontecimentos

Um JavaScript pode ser executado quando ocorre um evento, como quando um usuário clica em um elemento HTML. Para executar código quando um usuário clica em um elemento, adicione o código JavaScript para um atributo de evento HTML:

```
onclick=JavaScript
```

Exemplos de eventos HTML:

- Quando um usuário clica com o mouse
- Quando uma página web foi carregada
- Quando uma imagem tiver sido carregado
- Quando o mouse se move sobre um elemento
- Quando um campo de entrada (input) é alterado
- Quando um formulário HTML é submetido
- Quando um usuário aperta uma tecla

Neste exemplo, o conteúdo do elemento <h1> é alterada quando um usuário clica sobre ele:

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>
</body>
</html>
```

Neste exemplo, uma função é chamada por eventos:

```
<!DOCTYPE html>
<html>
<head>
<script>
function changetext(id)
{
id.innerHTML="Ooops!";
}
</script>
</head>
<body>
<h1 onclick="changetext(this)">Click on this text!</h1>
</body>
</html>
```

22.2. Atributos de eventos HTML

Para atribuir eventos a elementos HTML que você pode usar os atributos de eventos.

Atribuir um evento onclick de um elemento de botão:

```
<button onclick="displayDate()">Try it</button>
```

No exemplo acima, uma função chamada DisplayDate será executada quando o botão é clicado.

22.3. Atribuir eventos usando o DOM HTML

O HTML DOM permite atribuir eventos a elementos HTML usando JavaScript:

Atribuir um evento onclick de um elemento de botão:

```
<script>
document.getElementById("myBtn").onclick=function(){displayDate()};
</script>
```

No exemplo acima, uma função chamada DisplayDate é atribuído a um elemento HTML com o id = myButn ". A função será executada quando o botão é clicado.

22.4. Os eventos onload e onunload

Os eventos onload e onunload são acionados quando o usuário entra ou sai da página. O evento onload pode ser usado para verificar o tipo de navegador do visitante e versão do navegador e carregar a versão correta da página web com base na informação. Os eventos onload e onunload pode ser usado também para lidar com cookies.

```
<body onload="checkCookies()">
```

22.5. O evento onchange

O evento onchange são frequentemente usados em combinação com a validação de campos de entrada. Abaixo está um exemplo de como usar o onchange. A função upperCase() será chamada quando um usuário altera o conteúdo do campo de entrada.

```
<input type="text" id="fname" onchange="upperCase()">
```

22.6. Os eventos onmouseover e onmouseout

Os eventos onmouseover e onmouseout pode ser usado para acionar uma função quando o usuário passa o mouse sobre, ou fora de, um elemento HTML.

Um exemplo simples onmouseover, onmouseout:

```
<!DOCTYPE html>
<html>
<body>
```

```
<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-
color:#D94A38;width:120px;height:20px;padding:40px;">Mouse Over
Me</div>

<script>
function mOver(obj)
{
obj.innerHTML="Thank You"
}

function mOut(obj)
{
obj.innerHTML="Mouse Over Me"
}
</script>

</body>
</html>
```

22.7. Os eventos onmousedown, onMouseUp e onclick

O onmousedown, onmouseup e eventos onclick são todos partes de um clique do mouse. Primeiro, quando um botão do mouse é clicado, o evento onmousedown é acionado, então, quando o botão do mouse é liberado, o evento onmouseup é acionado, finalmente, quando o clique do mouse estiver concluída, o evento onclick é disparado.

Um exemplo simples onmousedown-onmouseup:

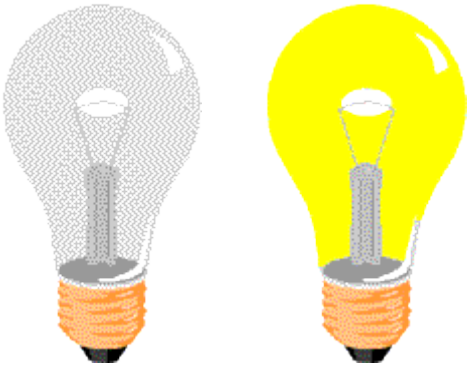
```
<!DOCTYPE html>
<html>
<body>

<div onmousedown="mDown(this)" onmouseup="mUp(this)"
style="background-
color:#D94A38;width:90px;height:20px;padding:40px;">Click Me</div>

<script>
function mDown(obj)
{
obj.style.backgroundColor="#1ec5e5";
obj.innerHTML="Release Me"
}

function mUp(obj)
{
obj.style.backgroundColor="#D94A38";
obj.innerHTML="Thank You"
}
</script>

</body>
</html>
```



Desligado Ligado

Esse exemplo mantém a lâmpada acessa ao segurar o botão do mouse em cima da imagem:

```
<!DOCTYPE html>
<html>
<head>
<script>
function lighton()
{
document.getElementById('myimage').src="bulbon.gif";
}
function lightoff()
{
document.getElementById('myimage').src="bulboff.gif";
}
</script>
</head>

<body>
<img id="myimage" onmousedown="lighton()" onmouseup="lightoff()"
src="bulboff.gif" width="100" height="180" />
<p>Click and hold to turn on the light!</p>
</body>
</html>
```

Este outro exemplo troca a cor do texto quando passamos o mouse por cima:

```
<!DOCTYPE html>
<html>
<body>

<h1 onmouseover="style.color='red'"
onmouseout="style.color='black'">
Passe o mouse sobre esse texto</h1>

</body>
</html>
```

Mais um exemplo: ao clicar sobre um caixa de entrada de texto, ela muda de cor:

```
<!DOCTYPE html>
<html>
```

```
<head>
<script>
function myFunction(x)
{
x.style.background="yellow";
}
</script>
</head>
<body>

Enter your name: <input type="text" onfocus="myFunction(this)">

<p>When the input field gets focus, a function is triggered which
changes the background-color.</p>

</body>
</html>
```

Por fim, este exemplo demonstra a execução do evento onload:

```
<!DOCTYPE html>
<html>
<head>

<script>
function mymessage()
{
alert("This message was triggered from the onload event");
}
</script>
</head>

<body onload="mymessage()">
</body>

</html>
```

23. DOM: adicionando e removendo nós (Elementos)

23.1. Criação de novos elementos HTML

Para adicionar um novo elemento para o HTML DOM, você deve criar o elemento (nó de elemento) primeiro, e depois anexá-lo a um elemento existente.

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para=document.createElement("p");
var node=document.createTextNode("Este é novo.");
para.appendChild(node);

var element=document.getElementById("div1");
element.appendChild(para);
</script>
```

Exemplo Explicado

Este código cria um novo elemento <p>:

```
var para=document.createElement("p");
```

Para adicionar texto ao elemento <p>, você deve criar um nó de texto primeiro. Este código cria um nó de texto:

```
var node=document.createTextNode("This is a new paragraph.");
```

Em seguida, você deve anexar o nó de texto para o elemento <p>:

```
para.appendChild(node);
```

Finalmente, você deve anexar o novo elemento a um elemento existente.

Este código encontra um elemento existente:

```
var element=document.getElementById("div1");
```

Este código adiciona o novo elemento ao elemento existente:

```
element.appendChild(para);
```

23.2. Remoção de elementos HTML existente

Para remover um elemento HTML, você deve conhecer o pai do elemento:

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<script>
var parent=document.getElementById("div1");
```

```
var child=document.getElementById("p1");
parent.removeChild(child);
</script>
```

Exemplo Explicado

Este documento HTML contém um elemento com dois nós filhos (dois elementos <p>):

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
```

Encontre o elemento com id = "div1":

```
var parent=document.getElementById("div1");
```

Localize o elemento <p> com id = "p1":

```
var child=document.getElementById("p1");
```

Remover o elemento filho do pai:

```
parent.removeChild(child);
```

Seria bom para ser capaz de remover um elemento sem se referir ao pai, mas desculpe. O DOM precisa saber tanto o elemento que você deseja remover, e seu pai. Aqui está uma solução comum: encontrar o filho que você deseja remover, e usar sua propriedade parentNode para encontrar o pai:

```
var child=document.getElementById("p1");
child.parentNode.removeChild(child);
```