Trabalhando com objetos em JavaScript

Sumário

Capítulo 1:
Visão Geral

Objetivo

Ao final deste capítulo você será capaz de:

- Entender os fundamentos da Linguagem JavaScript;
- Construir programas básicos em JavaScript utilizando instruções de manipulação de objetos.

O que é JavaScript

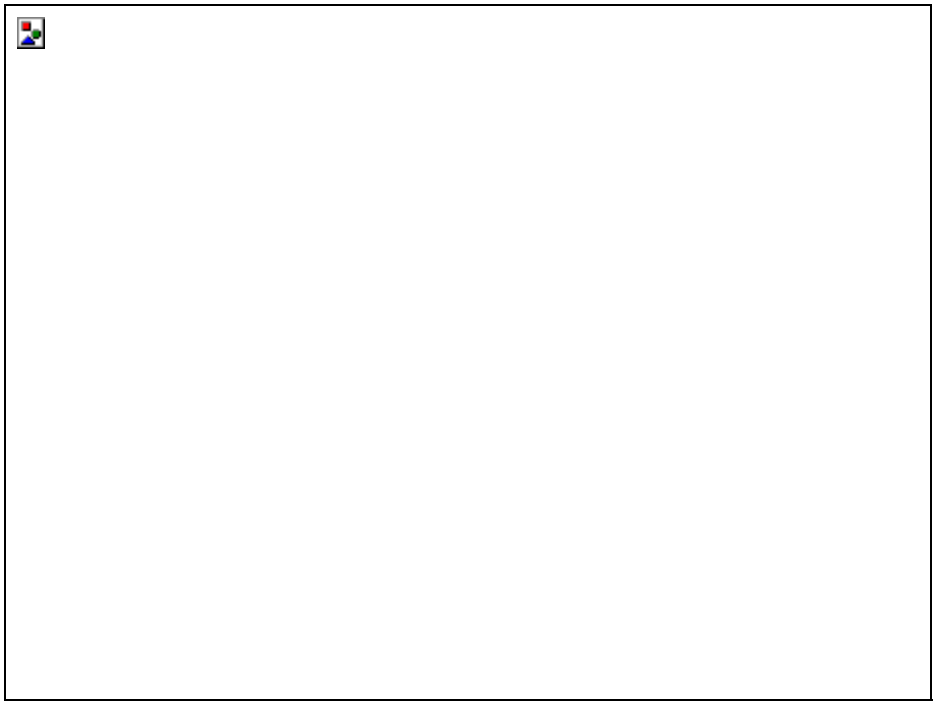
JavaScript é uma linguagem orientada à objeto criada pela Netscape. Sua funcionalidade essencial implementa um conjunto de objetos tais como Array, Date e Math e um conjunto de elemento como operadores estruturas de controle e instruções.

JavaScript pode ser estendido à uma série de propósitos tais como:

- Client-Server JavaScript: Estende as funcionalidades essenciais fornecendo objetos para controlar o navegador e o DOM.
- Server-Side JavaScript: Extenue as funcionalidades essenciais fornecendo objetos que permitem ao JavaScript ser executado no servidor.

JavaScript permite ainda a criação de páginas dinâmicas (DHTML) que processa entradas do usuário, cria animações, etc.

CS (Client-Side) e SS(Server-Side) convivem juntas e tem um núcleo em comum. A figura abaixo ilustra este núcleo.



Valores e variáveis

JavaScript reconhece 4 tipos de valores: Números, lógicos, strings, null e undefined. `Null` é uma palavra-chave especial simbolizando um valor nulo. JavaScript assim como java é Case-sensitive. Portanto `null` é diferente de `Null` ou de `NULL`. Undefined também é uma palavra-chave que representa um valor indefinido.

A conversão entre estes tipos de dados é dinâmica. Você não tem que especificar o tipo de dado quando declara uma variável. A conversão é feita durante a execução do script.

```
var answer = 42;
```

O tipo de dado pode ser mudado sem causar nenhum erro:

```
answer = "Conteudo da variavel";
```

Declaração de variáveis

Você pode declara variáveis de duas maneiras:

- Atribuindo um valor. Por exemplo x = 42;
- Utilizando a palavra var. Por exemplo var x = 42.

Todas as variáveis que não tiveram atribuição de valores possui o valor undefined.

Anotações

Expressões

Uma expressão operadores, variáveis e expressões que avaliam um determinado valor.

Conceitualmente existem dois tipos de expressões: aquelas que atribuem valor à uma variável e aquelas que simplesmente possuem um valor. Por exemplo, a expressão x = 7 é uma expressão que atribui 7 à variável x. Esta expressão usa operadores de atribuição. Por outro lado a expressão 3 + 4 não atribui nenhum valor a nenhuma variável. Estes são chamados simplesmente de operadores.

JavaScript possui os seguintes tipos de expressões:

- Aritméticas: avalia um número, por exemplo 3,14159
- De caracteres: avalia string de caracteres como “Fred”
- Lógicas: avalia true ou false

Anotações

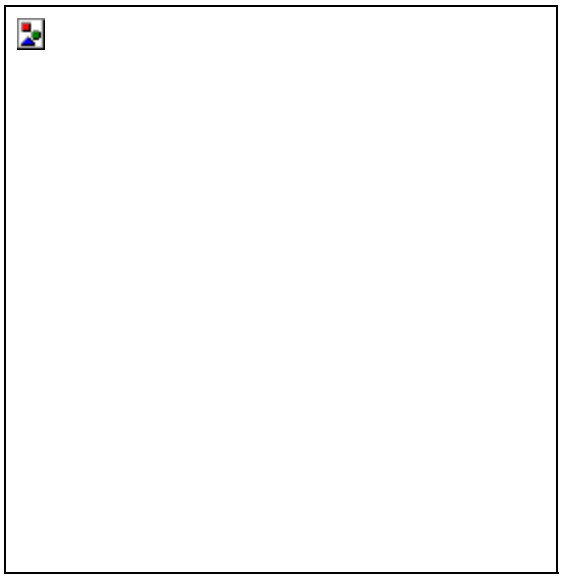
Operadores

JavaScript possui os seguintes tipos de operadores:

- de atribuição;
- de comparação;
- aritméticos;
- bitwise;
- lógicos;
- string;
- especiais.

Citaremos aqui apenas os mais utilizados.

A tabela abaixo mostra todos os operadores de atribuição:



Anotações

Esta tabela mostra os operadores de comparação:

Operator	Description	Examples returning true[a]

Equal (==)	Returns true if the operands are equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	`3 == var1`
		`"3" == var1`
		`3 == '3'`

Not equal (!=)	Returns true if the operands are not equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	`var1 != 4` `var2 != "3"`
Strict equal (===)	Returns true if the operands are equal and of the same type.	`3 === var1`
Strict not equal (!==)	Returns true if the operands are not equal and/or not of the same type.	`var1 !== "3"` `3 !== '3'`
Greater than (>)	Returns true if the left operand is greater than the right operand.	`var2 > var1`
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	`var2 >= var1` `var1 >= 3`
Less than (<)	Returns true if the left operand is less than the right operand.	`var1 < var2`
Less than or equal (<=)	Returns true if the left operand is less than or equal to the right operand.	`var1 <= var2` `var2 <= 5`

Anotações

Operadores aritméticos:

Operator	Description	Example
% (Modulus)	Binary operator. Returns the integer remainder of dividing the two operands.	12 % 5 returns 2.
++ (Increment)	Unary operator. Adds one to its operand. If used as a prefix operator (++x), returns the value of its operand after adding one; if used as a postfix operator (x++), returns the value of its operand before adding one.	If x is 3, then ++x sets x to 4 and returns 4, whereas x++ sets x to 4 and returns 3.
-- (Decrement)	Unary operator. Subtracts one to its operand. The return value is analogous to that for the increment operator.	If x is 3, then --x sets x to 2 and returns 2, whereas x++ sets x to 2 and returns 3.
- (Unary negation)	Unary operator. Returns the negation of its operand.	If x is 3, then -x returns -3.

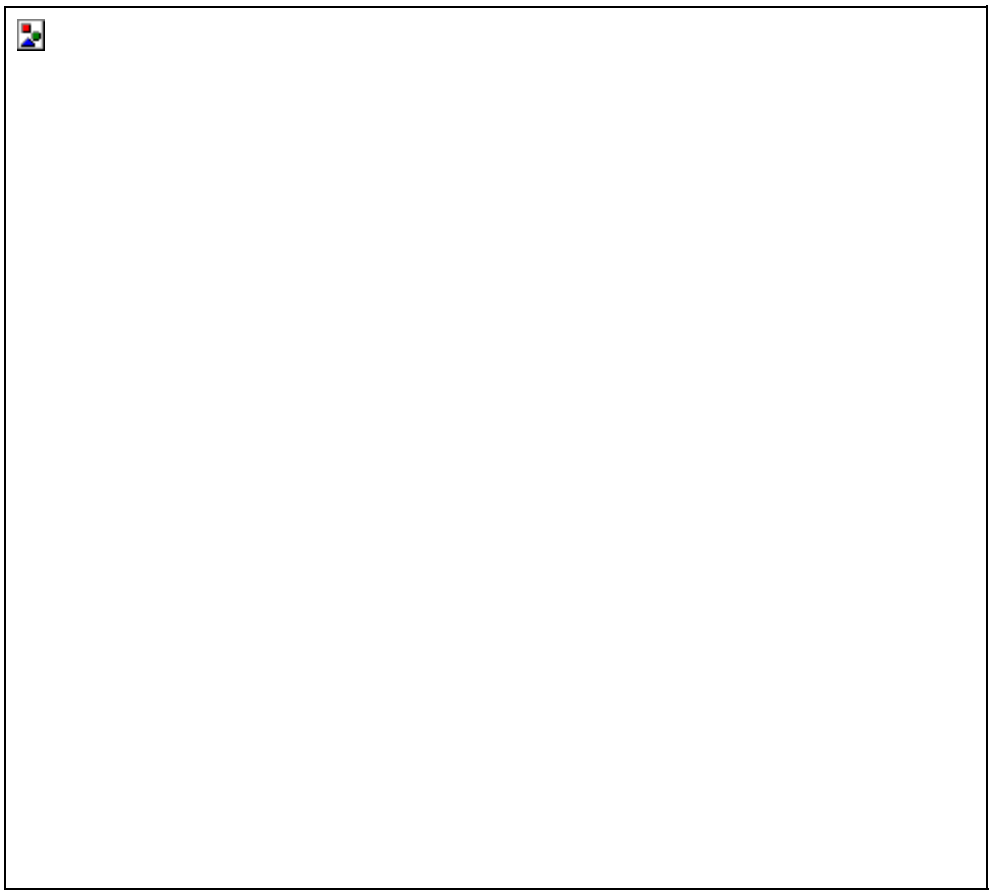
Operadores lógicos:

Operator	Usage	Description
&&	`expr1 && expr2`	(Logical AND) Returns `expr1` if it can be converted to false; otherwise, returns `expr2`. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.
||	`expr1 || expr2`	(Logical OR) Returns `expr1` if it can be converted to true; otherwise, returns `expr2`. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns false.
!	`!expr`	(Logical NOT) Returns false if its single operand can be converted to true; otherwise, returns true.

Anotações

Precedência de operadores

A tabela abaixo mostra a precedência de operadores, da mais baixa para mais alta.



Anotações

Instruções de manipulação de objetos

Existem duas instruções de manipulação de objetos: for... in e with.

for... in

A instrução for... in varre o vetor de propriedades do objeto especificado. No exemplo abaixo a função `dump_props` recebe como argumento um objeto e o nome do objeto. Então ele varre todas propriedades retornando o nome e o valor de cada propriedade:

```
function dump_props (obj, obj_name) {
   var result = "";
   for (var i in obj) {
      result += obj_name + "." + i + "=" + obj[i] + "<BR>"
   }
   return result;
}
```

with

A instrução with estabelece um objeto padrão ao qual o conjunto de propriedades utilizadas na seqüência da declaração pertence. No exemplo abaixo a instrução with se refere à propriedade PI e aos métodos cos e sin, sem especificar novamente o nome do objeto:

```
with (Math) {
   a = PI * r * r;
   x = r * cos(PI);
   y = r * sin(PI/2);
}
```

Anotações

Funções

Funções é um dos módulos fundamentais do JavaScript. O JavaScript assim como outras linguagens fornecem funções predefinidas para realização de tarefas básicas. São estas:

- eval: avalia uma string executando as instruções contidas nesta string;
- isFinite: avalia se um determinado número é finito ou não;
- isNaN: Avalia um argumento e determina se ele não é um número;
- parseInt e parseFloat: Converte um valor de caracter para inteiro ou ponto flutuante;
- Number e String: Converte um objeto para número ou caracter;

Anotações

Revisão

1. Utilizando o exemplo da apostila construa uma função que mostre na tela todas propriedades de um formulário.

2. Quais são os tipos de variáveis existentes em JavaScript?

Capítulo 2:
Trabalhando com Objetos

Objetivo

Ao final deste capítulo você será capaz de:

- Criar seus próprios objetos
- Criar propriedades e métodos para estes objetos
- Utilizar os objetos predefinidos pelo JavaScript

Objetos e Propriedades

Para acessar as propriedades dos objetos utiliza-se a seguinte notação:

NomedoObjeto.NomedaPropriedade

Ambos são case sensitive. Para definir a propriedade basta atribuir o valor. Por exemplo, suponhamos que exista um objeto chamado `myCar`. O seguinte código atribui valores às propriedades make, model, year:

```
myCar.make = "Ford";
myCar.model = "Mustang";
myCar.year = 1969;
```

Um vetor é um conjunto ordenado de valores associado à nomes de variáveis. Propriedades e vetores estão intimamente ligados em JavaScript; Na verdade eles são interfaces diferentes para o mesma estrutura de dados. Por exemplo, as mesmas propriedades acima poderiam ser criadas da seguinte maneira:

```
myCar["make"] = "Ford";
myCar["model"] = "Mustang";
myCar["year"] = 1967;
```

Anotações

Este tipo de vetor é chamado vetor associativo pois cada elemento do índice está associado à um valor string. O exemplo abaixo ilustra este funcionamento.

```
function show_props(obj, obj_name) {
   var result = ""
   for (var i in obj)
   result += obj_name + "." + i + " = " + obj[i] + "\n"
   return result
}
```

Executando `show_props(myCar, "myCar")` retornaria o seguinte texto:

```
myCar.make = Ford
myCar.model = Mustang
myCar.year = 1967
```

Anotações

Criando novos objetos

Apesar do JavaScript fornecer inúmeros objetos, você pode criar seus próprios objetos. Os objetos podem ser criados utilizando uma função construtora ou sendo inicializado.

Inicializando objetos

Os objetos podem ser inicializados atribuindo valores diretamente junto com a declaração.

```
NomedoObjeto = {Propriedade1:valor1, Propriedade2:valor2, Propriedade3:valor3}
```

O exemplo abaixo cria o objeto myHonda com três propriedades. Note que o motor é um objeto:

```
myHonda = {color:"red", wheels:4, engine:{cylinders:4, size:2}}
```

Usando função construtora

Para utilizar função construtora são necessários os seguintes passos: 1)Definir o tipo do objeto escrevendo a função construtora. 2) Criar uma nova instância do objeto usando a palavra new.

O exemplo abaixo cria a função construtora:

```
function car(make, model, year) {
   this.make = make
   this.model = model
   this.year = year
}
```

E agora a função é chamada passando os parâmetros para alimentar o objeto:

```
mycar  = new car("Jaguar", "XR8", 1995)
```

Anotações

Indexando propriedades dos objetos

Em versões anteriores do JavaScript era possível se referir à um objeto através do nome da propriedade ou o número de índice. Atualmente se você define uma propriedade utilizando o nome, você sempre utilizará o nome e o mesmo vale para número de índice.

A exceção acontece em objetos do HTML, como o objeto forms. Estes sempre podem ser referenciados utilizando ambos meios de acesso. Por exemplo, se o segundo formulário de um documento chama-se myForm, este pode ser acessado utilizando a sintaxe `document.forms[1]` (já que o índice começa do zero) ou `document.forms["myForm"]` ou ainda `document.myForm`.

Definindo métodos

Um método é uma função associada à um objeto. Você define métodos da mesma maneira que define funções:

```
Objeto.NomeMetodo = NomeFuncao
```

Utilizando o princípio do exemplo anterior, para adicionar o método displayCar() ao objeto car primeiro declara-se o método:

```
function displayCar() {
   var result = "A Beautiful " + this.year + " " + this.make
   + " " + this.model
   document.write(result)
}
```

Para fazer desta função um método você a atribui usando o seguinte código:

```
this.displayCar = displayCar;
```

Anotações

Para chamar o método você utiliza o seguinte comando:

```
car1.displayCar();
```

Usando *this* para referências à objetos

JavaScript tem uma palavra especial, this, que pode ser usada dentro de um método para referenciar o objeto corrente. Por exemplo, suponha que você tenha uma função chamada validate que valida o valor da propriedade do objeto:

```
function validate(obj, lowval, hival) {
   if ((obj.value < lowval) || (obj.value > hival))
   alert("Invalid Value!")
}
```

Então você chama a função validate no evento onChange em cada campo do formulário:

```
<INPUT TYPE="text" NAME="age" SIZE=3 onChange="validate(this, 18, 99)">
```

Desta maneira, this no teste referencia cada campo do formulário.

Apagando objetos

Para remover objetos da memória utilize a instrução delete de acordo com o exemplo:

```
myobj=new Number()
delete myobj // removes the object and returns true
```

Anotações

Objetos predefinidos

JavaScript possui alguns objetos predefinidos que podem ser utilizados na criação de novos objetos

Array

Como JavaScript não tem um tipo de dado do tipo array você pode utilizar este objeto para criar variáveis do tipo array:

```
coffees = ["French Roast", "Columbian", "Kona"]
```

Date

Este objeto pode ser utilizados para manipulações de valores em formato de data como no exemplo abaixo:

```
function JSClock() {
   var time = new Date()
   var hour = time.getHours()
   var minute = time.getMinutes()
   var second = time.getSeconds()
   var temp = "" + ((hour > 12) ? hour - 12 : hour)
   temp += ((minute < 10) ? ":0" : ":") + minute
   temp += ((second < 10) ? ":0" : ":") + second
   temp += (hour >= 12) ? " P.M." : " A.M."
   return temp
}
```

Anotações

Function

Seu uso sugere uma alternativa à maneira tradicional de criar funções. Pode criar funções utilizando apenas uma instrução, como no exemplo:

```
functionObjectName = new Function ([arg1, arg2, ... argn], functionBody)
```

Math

Fornece alguns métodos interessantes para manipulação de valores matemáticos: `abs, sin, cos, tan, acos, asin, atan, exp, log, ceil, floor, min, max, pow, round, sqrt`

Number

Possui algumas propriedades úteis para avaliar o valor do número: `MAX_VALUE, MIN_VALUE, NaN, NEGATIVE_INFINITY, POSITIVE_INFINITY.`

String

Manipula os caracteres contidos em uma string.

Para maiores informações sobre funções consulte o JavaScript Reference.

Anotações

Revisão

1. Crie um objeto através do método de função construtora e associe a ele propriedades e métodos. Utilize como tema a palavra aluno.

2. Crie uma função que retorne os minutos da hora atual.

Capítulo 3:
Usando Objetos do Navegador

Objetivo

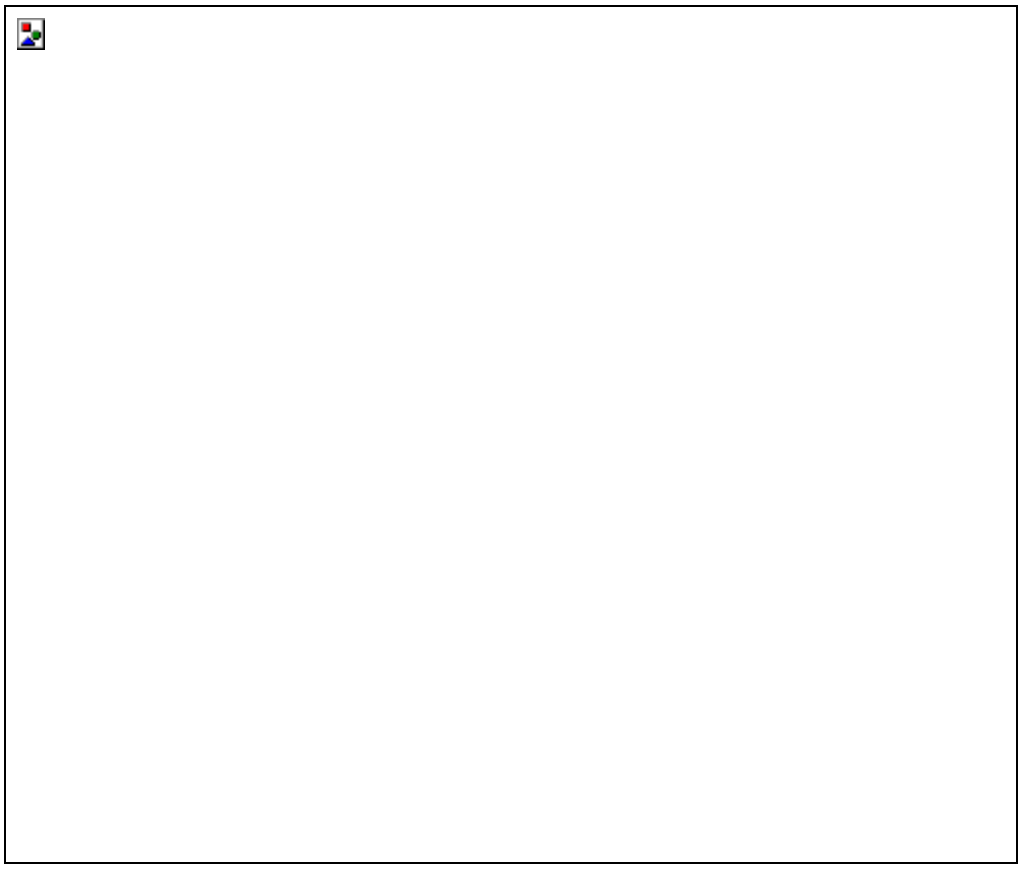
Ao final deste capítulo você será capaz de :

- Utilizar os objetos HTML para criar scripts;
- Definir a melhor maneira de acessar um script

Hierarquia de objetos

Quando um documento é carregado, o navegador cria um número de Objetos e propriedades baseadas no documento HTML e outras informações pertinentes. Estes objetos existem em uma hierarquia que reflete a estrutura da página HTML. À esta estrutura chamamos de DOM (Document Objet Model).

Apesar do esforço do W3C para padronizar o DOM dos navegadores, ainda existe uma infinidade de divergências entre o DOM utilizado pelo Netscape e pelo Internet Explorer. A figura abaixo ilustra a parte em comum entre os dois navegadores:



Anotações

Nesta hierarquia, o objeto descendente é propriedade do objeto ascendente. Por exemplo, um formulário chamado form1 é um objeto e propriedade do objeto document e é referenciado como document.form1. Para uma lista de todos os objetos, propriedades, métodos e eventos leia o JavaScript Reference.

Cada página possui os seguintes objetos:

- navigator
- window
- document
- location
- history

Anotações

Principais Objetos

Abaixo segue a descrição dos principais objetos HTML que podem ser manipulados pelo JavaScript:

Window e frame

O objeto window é o objeto pai de todos os outros objetos. Cada objeto window tem inúmeros objetos interessantes tais como: open, para abrir uma nova janela; close, para fechar; alert, para enviar uma mensagem para o usuário; confirm, para fazer uma pergunta ao usuário; prompt, para que o usuário entre com algum valor.

Document

O objeto document é o objeto mais usado em JavaScript. Para maiores informações consulte o JavaScript Reference

Form

Cada formulário no documento cria um objeto form. Como um documento pode conter mais de um formulário, estes são armazenados no vetor chamado forms. Cada elemento dentro do objeto form é armazenado em um vetor chamado elements. Assim todos eles podem ser acessados através do seu próprio nome como índice:

```
Document.forms[0].elements[0]
```

Location

O objeto location possui propriedades baseadas no endereço corrente. Por exemplo a propriedade hostname retorna o nome do servidor que hospeda a página carregada. O objeto location possui também dois métodos: reload, que força o navegador a recarregar a página e replace que carrega a URL especificada.

Anotações

History

O objeto history contém uma lista de caracteres representado as URLs que o usuário já visitou. Você pode acessar todas as URLs navegadas pelo usuário através das propriedades current, next e previous. Você também pode utilizar o vetor history para acessar outras ocorrências.

Você também pode utilizar o método go que recebe como parâmetro o número da página para onde o navegador deve ir. No exemplo abaixo o script manda o navegador para duas páginas anteriores à página atual:

```
history.go(-2)
```

Navigator

O objeto Navigator contém informações sobre a versão do navegador utilizado. O objeto navegador possui três métodos:

- javaEnabled: Testa se o navegador está habilitado para java;
- preference: permite o uso de scripts para alterar as preferências do usuário;

Anotações

Revisão

1. Crie um formulário de envio de e-mail.

2. Crie um script de validação para validar se o usuário preencheu ou não o e-mail no campo e-mail. Utilize seus conhecimento e os conhecimento adquiridos neste curso.