



# JavaScript

**Prof. David Fernandes de Oliveira**  
**Instituto de Computação**  
**UFAM**

# O que é JavaScript

- Linguagem de **script** criada pela Netscape em 1995, interpretada e de alto nível
- Possui **tipagem dinâmica** e é baseada em objetos
- Não possui conexão com Java, a não ser pela sintaxe
- Atualmente é a principal linguagem para **programação client-side** em browsers

# Onde colocamos o JavaScript?

- Embarcado em documentos HTML entre as marcações `<script>` e `</script>`

```
<script language="javascript">
```

Comandos JavaScript vão aqui!

```
</script>
```

- Em arquivos externos, carregados usando

```
<script src="program.js" ...></script>
```



Arquivo JavaScript

Nada entre as tags

# Onde colocamos o `<script>`?

- **No cabeçalho do documento HTML**
  - Neste caso ele é lido antes do documento HTML ser renderizado. O código será executado imediatamente.
- **In the body of the HTML document**
  - Neste caso ele é lido enquanto o documento HTML é renderizado. Quando encontra o `<script>`, o browser para a renderização para interpretar o código JavaScript.

# Escrevendo HTML com JavaScript

- Mais a frente falaremos sobre o **Document Object Model (DOM)**, que permite que o JavaScript interaja com o documento HTML
- Por enquanto iremos usar o `document.write()` para produzir uma saída HTML usando código JavaScript
- Exemplo:

```
document.write("<h1>Hello World</h1>");
```

# Programa Hello World (1)

```
<html>
<head>
<title> ... </title>
<script language="javascript">
    document.writeln("<h1>Hello World</h1>");
</script>
</head>
<body>
</body>
</html>
```

Qualquer coisa aqui irá ser  
mostrada depois de Hello World

<examples/simple/HelloWorld1.html>

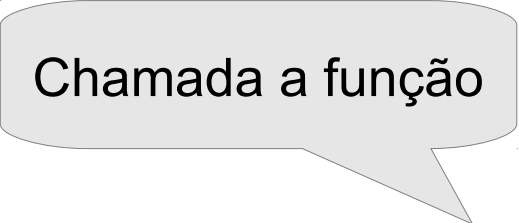
# Programa Hello World (2)

```
<html>
<head>
<title> ... </title>
</head>
<body>
<script language="javascript">
    document.writeln("<h1>Hello World</h1>");
</script>
</body>
</html>
```

<examples/simple/HelloWorld2.html>

# Programa Hello World (3)

```
<html><head><title> ... </title>
<script language="javascript">
function hello () {
    document.write("<h1>Hello World</h1>");
}
</script>
</head>
<body>
<script language="javascript">hello();</script>
</body>
</html>
```



Chamada a função

<examples/simple/HelloWorld3.html>



# Programa Hello World (4)

```
<html><head><title> ... </title>
<script language="javascript">
function hello(nome) {
    document.write("<h1>Hello, " + nome + "</h1>");
}
</script>
</head>
<body>
<script language="javascript">hello("UFAM");</script>
</body>
</html>
```

Chamada a função com parâmetros

<examples/simple/HelloWorld4.html>

# Programa Hello World (5)

```
<html><head><title> ... </title>
<script language="javascript">
function hello(nome) {
    return("<h1>Hello, " + nome + "</h1>");
}
</script>
</head>
<body>
<script language="javascript">
    document.writeln(hello("UFAM"));
</script>
</body>
</html>
```



Retorno de função

<examples/simple/HelloWorld5.html>

# Programa Hello World (6)

```
<html><head><title> ... </title>
<script language="javascript" src="hello.js">
</script>
</head>
<body>
<script language="javascript">hello();</script>
</body>
</html>
```

examples/simple/HelloWorld6.html

```
function hello()
{
    document.writeln("<h1>Hello World</h1>");
}
```



hello.js

# Programa Hello World (7)

```
<html><head><title> ... </title>
<script language="javascript">
    window.alert("Hello World");
</script>
</head>
<body>
```



Caixa de alerta

Feché a caixa de alerta para ver este texto. Use a função de reload para rodar o script novamente.

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

<examples/simple/HelloWorld7.html>

# Programa Hello World (8)

```
<html><head><title> ... </title>
<script language="javascript">
    function hello()
    { document.write("<h1>Hello World</h1>"); }
</script>
</head>
<body onload="hello()">
Este texto nunca será visto.
</body>
</html>
```

Função carregada sobre  
o evento onload

<examples/simple/HelloWorld8.html>

# Variáveis JavaScript

- JavaScript possui tipagem dinâmica: tipos são associados a valores, não a variáveis
- A palavra **var** é usada para definir variáveis:

```
var i = 1;
```

```
i = "a string";
```

```
i = new Date();
```

Tipos não são informados

- JavaScript é case-sensitive

# Variáveis JavaScript

- Variáveis definidas dentro de uma função são chamadas de **variáveis locais**
  - Essas variáveis só podem ser acessadas dentro de suas próprias funções
- Variáveis definidas diretamente no elemento script são chamadas **variáveis globais**
  - Elas podem ser acessadas de qualquer lugar, incluindo outros scripts

# Variáveis JavaScript

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <script type="text/javascript">
      var myGlobalVar = "apples";

      function myFunc(name) {
        var myLocalVar = "sunny";
        return ("Hello " + name + ". Today is " + myLocalVar + ".");
      };
      document.writeln(myFunc("Adam"));
    </script>
    <script type="text/javascript">
      document.writeln("I like " + myGlobalVar);
    </script>
  </body>
</html>
```



# Tipos de variáveis

- **Tipo número**
  - O tipo número é usado para representar números inteiros e ponto flutuante
- **Tipos string**
  - Colocados entre aspas simples ou duplas
- **Tipos undefined**
  - Variáveis criadas mas não inicializadas
- **Tipos Array, Object, Booleano ...**

# Tipo Numérico

- Número é um tipo objeto que representa inteiros e números de ponto flutuante
  - O tipo numérico (inteiro, real) é definido pelo valor assumido pela variável
- Não existe um tipo inteiro separado
- +, -, \*, / são operações numéricas

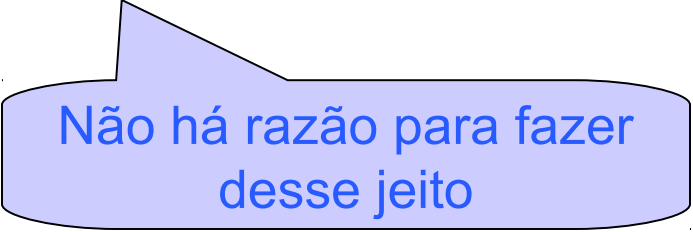
```
<script type="text/javascript">  
    var daysInWeek = 7;  
    var pi = 3.14;  
    var hexValue = 0xFFFF;  
</script>
```

# Valores Numéricos Especiais

- **Infinity**
- **NaN**
  - Not a number
- **Number.MAX\_VALUE**
  - Maior número positivo
- **Number.MIN\_VALUE**
  - Menor número positivo maior que zero

# Tipo String (1)

- Objetos **String** são usados para manipular um pedaço de texto armazenado
- Usam aspas simples ou duplas como delimitadores
- As seguintes declarações são iguais
  - `var s = "hello";`
  - `var s = new String("hello");`



Não há razão para fazer  
desse jeito

# Tipo String (2)

- Propriedade:
  - `s.length`
- Métodos :
  - `s.substring(i)`
  - `s.substring(i, j)`
  - `s.charAt(i)`
  - `s.indexOf(pattern)`

# Tipos Objetos (1)

- JavaScript suporta a noção de objetos

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <script type="text/javascript">
      var myData = new Object();
      myData.name = "Adam";
      myData.weather = "sunny";

      document.writeln("Hello " + myData.name + ". ");
      document.writeln("Today is " + myData.weather + ".");
    </script>
  </body>
</html>
```

# Tipos Objetos (2)

- Formato literal, para definição do objeto

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <script type="text/javascript">
      var myData = {
        name: "Adam",
        weather: "sunny"
      };

      document.writeln("Hello " + myData.name + ". ");
      document.writeln("Today is " + myData.weather + ".");
    </script>
  </body>
</html>
```

# Tipos Objetos (3)

- Definindo funções como métodos

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <script type="text/javascript">
      var myData = {
        name: "Adam",
        weather: "sunny",
        printMessages: function() {
          document.writeln("Hello " + this.name + ". ");
          document.writeln("Today is " + this.weather + ".");
        }
      };

      myData.printMessages();

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



# Tipos Objetos (4)

- Outra sintaxe para acesso e escrita das propriedades de um objeto:

```
myData["weather"] = "raining";
```

- Esta sintaxe pode ser bastante conveniente para acessar o valor de uma propriedade usando uma variável:

```
var propName = "weather";  
myData[propName] = "raining";
```

# Tipos Objetos (5)

- Podemos usar um loop `for...in` para acessar cada propriedade/método de um objeto

```
for (var prop in myData) {  
    document.writeln("Name: " + prop + " Value: " + myData[prop]);  
}
```

- Em cada iteração, a variável `prop` recebe o nome de uma das propriedades/métodos do objeto

# Tipos Objetos (6)

- Para apagar uma propriedade ou método de um objeto, podemos usar o operador **delete**

```
<script type="text/javascript">
    var myData = {
        name: "Adam",
        weather: "sunny",
    };

    myData.sayHello = function() {
        document.writeln("Hello");
    };

    delete myData.name;
    delete myData["weather"];
    delete myData.sayHello;
</script>
```

# Tipos Objetos (7)

- O operador **in** pode ser usado para verificar se um objeto possui uma dada propriedade ou método

```
<script type="text/javascript">
    var myData = {
        name: "Adam",
        weather: "sunny",
    };

    var hasName = "name" in myData;
    var hasDate = "date" in myData;

    document.writeln("HasName: " + hasName);
    document.writeln("HasDate: " + hasDate);

</script>
```

# Vetores (1)

- Vetores em JavaScript são um pouco diferentes de linguagens tradicionais

```
<script type="text/javascript">
```

```
var myArray = new Array();  
myArray[0] = 100;  
myArray[1] = "Adam";  
myArray[2] = true;
```

```
</script>
```

- O vetor cresce dinamicamente a medida que novos elementos são inseridos
- Os itens dos vetores podem ser de tipos diferentes

# Vetores (2)

- JavaScript também suporta criação de vetores no formato literal

```
<script type="text/javascript">  
    var myArray = [100, "Adam", true];  
    for (var i = 0; i < myArray.length; i++) {  
        document.writeln("Index " + i + ": " + myArray[i]);  
    }  
</script>
```

- Os elementos de um vetor podem ser acessados de forma similar a de outras linguagens

# Vetores (3)

- O objeto Array define um conjunto de métodos para manipulação de vetores

Method	Description	Returns
<code>concat(&lt;otherArray&gt;)</code>	Concatenates the contents of the array with the array specified by the argument. Multiple arrays can be specified.	Array
<code>join(&lt;separator&gt;)</code>	Joins all of the elements in the array to form a string. The argument specifies the character used to delimit the items.	string
<code>pop()</code>	Treats an array like a stack, and removes and returns the last item in the array.	object

# Vetores (3)

- O objeto Array define um conjunto de métodos para manipulação de vetores

Method	Description	Returns
push(<item>)	Treats an array like a stack, and appends the specified item to the array.	void
reverse()	Reverses the order of the items in the array in place.	Array
shift()	Like pop, but operates on the first element in the array.	object
slice(<start>,<end>)	Returns a sub-array.	Array
sort()	Sorts the items in the array in place.	Array
unshift(<item>)	Like push, but inserts the new element at the start of the array.	void



# Comentários

- Dois estilos de comentários

- `/* ..... */`

- `// .....`

# Declarações (1)

- Declarações **if** e **if-else**
- Possuem a mesma estrutura da linguagem C e Java
- Exemplo:

```
if (...)
{
    ...
}
else
{
    ...
}
```

# Declarações (2)

- Declarações **while**
- Mesma estrutura de C e Java
- Exemplo:

```
while (...)  
{  
    ...  
}
```

# Declarações (3)

- Declarações **do-while**
- Mesma estrutura de C e Java
- Exemplo:

```
do
{
    ...
} while (...);
```

# Declarações (4)

- Declarações **for**
- Mesma estrutura de C e Java
- Exemplo:

```
for ( . . . ; . . . ; . . . )  
{  
    . . .  
}
```

# Função Fatorial (1)

```
// Compute n!  
function factorial(n)  
{  
    var p = 1;  
    for (var k = 2; k <= n; k++)  
        p = p * k;  
    return p;  
}
```

# Função Fatorial (2)

```
function factorialTable()  
{  
    document.write("<pre>");  
    for (var k = 0; k <= 25; k++)  
    {  
        document.writeln(k + "! = " +  
            factorial(k));  
    }  
    document.write("</pre>");  
}
```

<examples/simple/factorial.html>

# Função para Inverter String (1)

```
// reverse the string s
function reverse(s)
{
    var sReverse = "";
    for (var k = 0; k < s.length; k++)
    {
        sReverse = s.charAt(k) + sReverse;
    }
    return sReverse;
}
```



# Função para Inverter String (2)

// Recursive version

```
function recursiveReverse(s)
{
    if (s.length <= 1) return s;
    return
        recursiveReverse(s.substring(1)) +
            s.charAt(0);
}
```

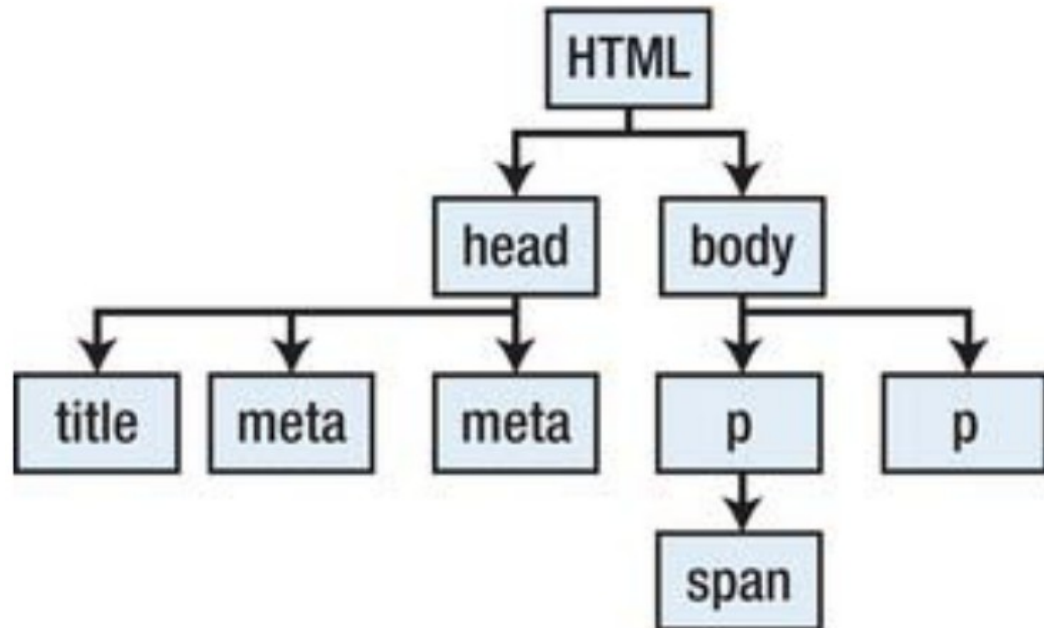
<examples/simple/reverse.html>

# Trabalhando com a DOM

- A **DOM (Document Object Model)** é usada para explorar e manipular o conteúdo de documentos HTML
- Neste momento da disciplina, iremos mostrar como:
  - Encontrar e alterar objetos JavaScript que representam elementos do documento HTML
  - Como responder às interações dos usuários através de eventos

# Trabalhando com a DOM

- A DOM é uma coleção de objetos JavaScript que representam os elementos de um documento HTML
- A DOM provê uma ponte entre a estrutura, o conteúdo e a apresentação de um documento HTML/CSS e o JavaScript



# Trabalhando com a DOM

- O acesso a DOM dá-se através do objeto **Document**, representado pela a variável global **document**
- O objeto Document provê informações sobre o documento como um todo, e dá acesso a objetos individuais da DOM
- A melhor forma de introduzir este objeto é através de um exemplo

# Exemplo de uso da DOM

```
<p id="icomp">
```

```
    O Instituto de Computação (IComp) é o mais novo instituto  
    da UFAM, tendo sido formado a partir do antigo Departamento  
    de Ciência da Computação (DCC).
```

```
</p>
```

```
<p id="atividades">
```

```
    Como toda unidade acadêmica, o IComp atua no ensino,  
    pesquisa e extensão, além de desempenhar atividades  
    administrativas.
```

```
</p>
```

```
<script>
```

```
    document.writeln("<pre>URL: " + document.URL);  
    var elems = document.getElementsByTagName("p");  
    for (var i = 0; i < elems.length; i++) {  
        document.writeln("Element ID: " + elems[i].id);  
        elems[i].style.border = "medium double black";  
        elems[i].style.padding = "4px";  
    }
```

```
    document.write("</pre>");
```

```
</script>
```

# Exemplo de uso da DOM

```
<p id="icomp">
```

O Instituto de Computação (IComp) é o mais novo instituto da UFAM, tendo sido formado a partir do antigo Departamento de Ciência da Computação (DCC).

```
</p>
```

```
<p id="atividades">
```

Como toda unidade acadêmica, o IComp atua no ensino, pesquisa e extensão, além de desempenhar atividades administrativas.

```
</p>
```

```
<script>
```

```
document.write(
```

```
var elems = doc
```

```
for (var i = 0;
```

```
document.write
```

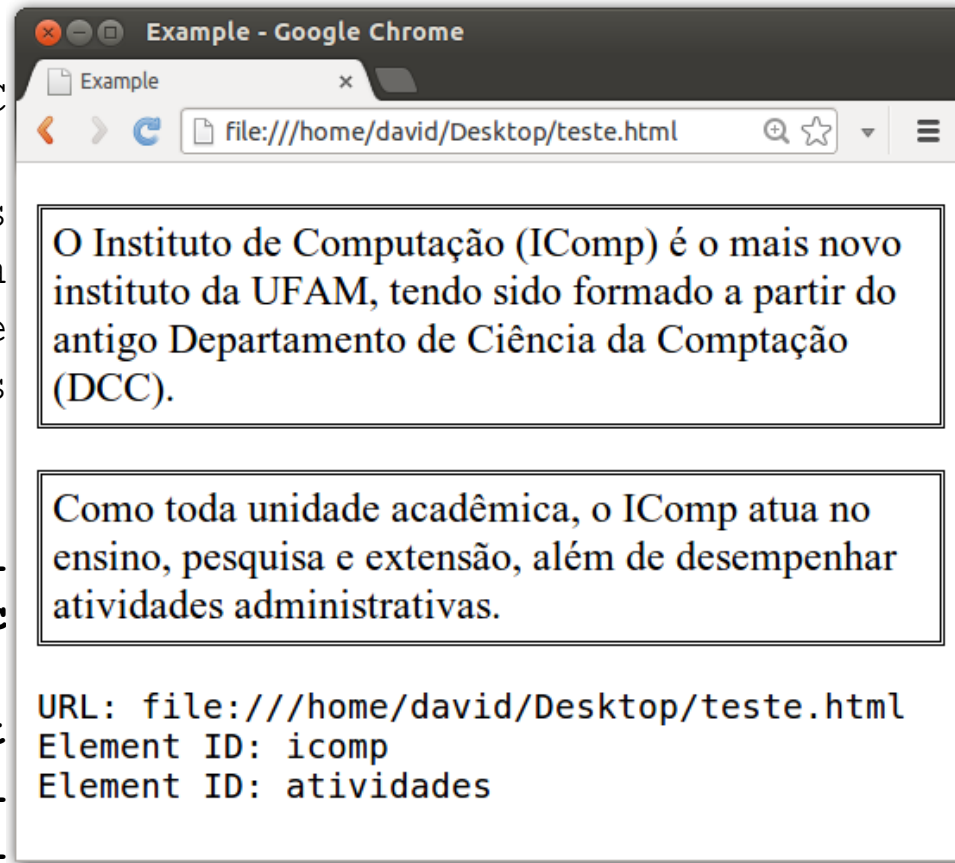
```
elems[i].style
```

```
elems[i].style
```

```
}
```

```
document.write("</pre>");
```

```
</script>
```

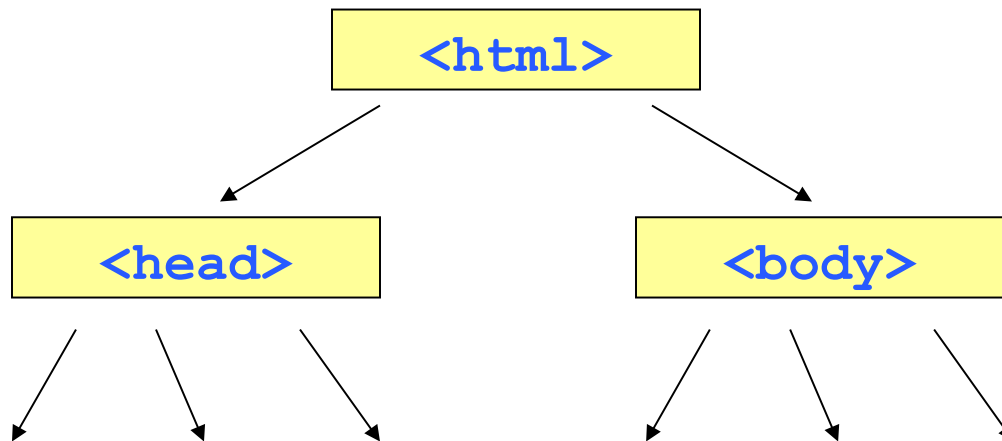


# Client-side JavaScript (1)

- Um browser define objetos para cada página, tais como o **window** e **document**, que podem interagir com o JavaScript
- A hierarquia de objetos provê acesso ao document object model (DOM)
- Existe também um modelo de eventos que associa eventos dos usuários a métodos JavaScript

# Client-side JavaScript (1)

- A combinação de JavaScript, DOM, e CSS é frequentemente chamada de **Dynamic HTML**
- Segunda a DOM, uma página Web possui a seguinte estrutura hierárquica:





# Hierarquia de objetos

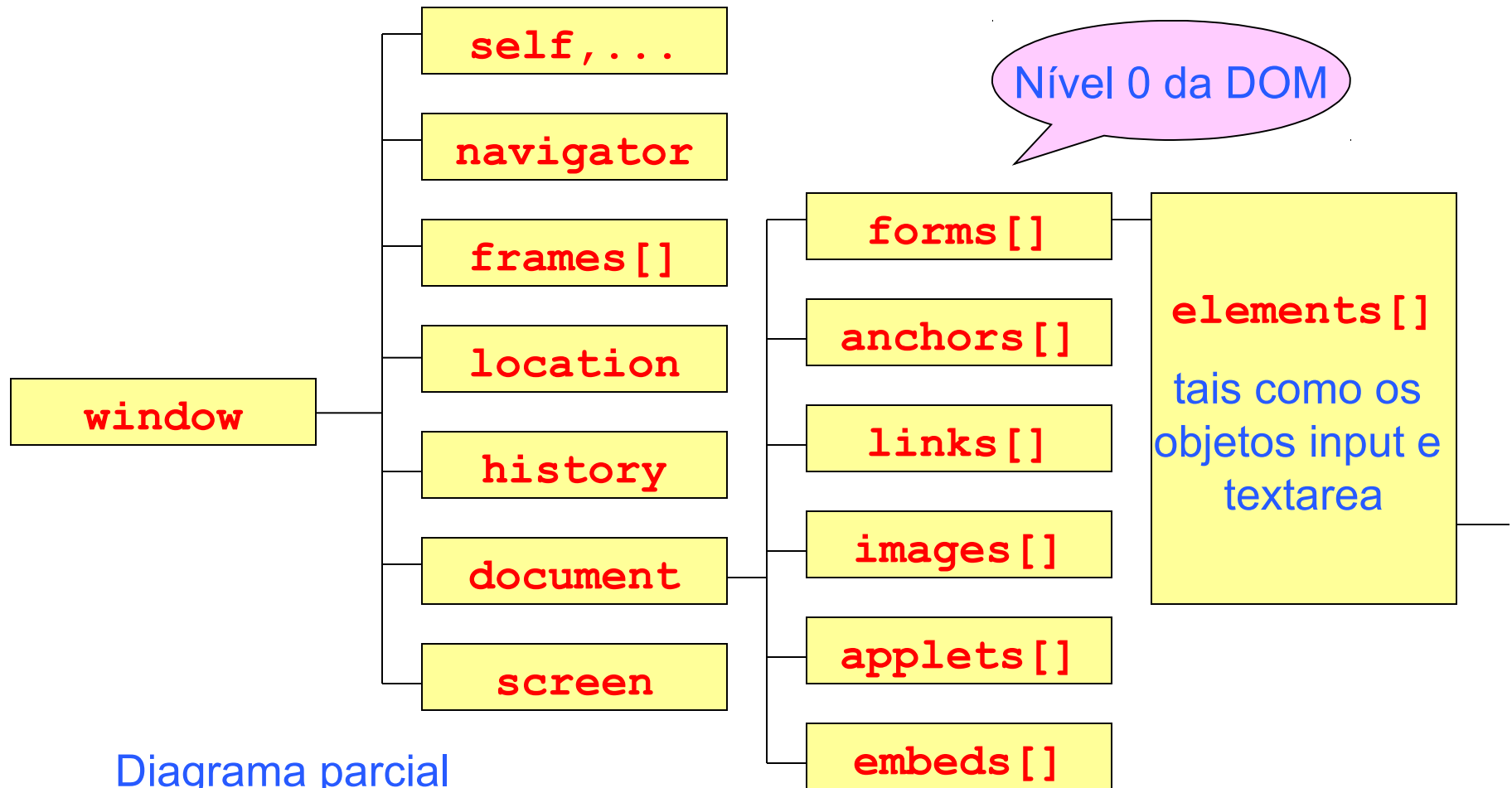


Diagrama parcial

# Alert, Confirm, Prompt (1)

- Interação com usuários usando caixa de diálogos
- `window.alert(msg) ;`
  - mostra uma mensagem em uma caixa que é fechada ao se clicar no botão OK
- `var r = window.confirm(msg) ;`
  - mostra uma mensagem junto com os botões OK e Cancel. Retorna true se o usuário clica em OK. Caso contrário retorna false

# Alert, Confirm, Prompt (2)

- `var r =`  
    `window.prompt(msg, default) ;`
  - mostra uma mensagem e um input de dados. O usuário entra com algum dado e então clica em OK, Clear, ou Cancel.
  - A string informada pelo usuário é retornada pelo método
  - Se o botão Cancel for clicado, o valor `null` é retornado

# Alert, Confirm, Prompt (3)

```
alert(  
  "This program can add two numbers\n" +  
  "You will be asked to enter two numbers\n" +  
  "Then you will be asked if you want to add them\n"  
);  
var first, second, num1, num2, sum;  
firstNumber = window.prompt(  
  "Enter 1st integer to add", "0");  
document.write("<br>", "You entered " + first);  
second = window.prompt(  
  "Enter 2nd integer to add", "0");  
document.write("<br>", "You entered " + second);
```

# Alert, Confirm, Prompt (4)

```
var ok = confirm(  
    "Do you want to add these two numbers?");  
if (ok)  
{  
    num1 = parseInt(first);  
    num2 = parseInt(second);  
    sum = num1 + num2;  
    document.write("<br>", "The sum of " +  
        first + " and " + second + " is " + sum);  
}
```

<examples/simple/dialog.html>

# Lendo Inputs de Formulários (1)

```
<form name="myForm">  
  <input name="myInput" type="text" value="">  
  ...  
</form>
```

- JavaScript pode referenciar o formulário usando
  - `document.myForm`
- O valor do elemento input pode ser referenciado usando
  - `document.myForm.myInput.value`

# Lendo Inputs de Formulários (2)

```
<form name="myForm">
```

```
...
```

```
<input name="myButton" type="button"  
      value="clickMe"  
      onclick="getInput()">
```

```
</form>
```

- Quando o botão é clicado, **getInput** é executado e ele pode referenciar os inputs do formulário usando
  - **document**.myForm.elementName

# Lendo Inputs de Formulários (2)

```
<form name="myForm">
```

```
...
```

```
<input name="myButton" type="button"  
      value="clickMe"  
      onclick="getInput(myForm)">
```

```
</form>
```

- Quando o botão é clicado, **getInput** é executado.
- Verifique que o nome do formulário pode ser passado como argumento



# Exemplo (1)

- Escreva um programa que calcula a área e a circunferência de um círculo:

Enter radius of circle	<input type="text" value="1"/>
<input type="button" value="Do calculations"/>	
Area	<input type="text"/>
Circumference	<input type="text"/>

# Exemplo (2)

use uma tabela  
para alinhar os  
elementos

```
<form name="myForm">
```

```
...
```

```
<input name="radiusField" type="text" value="1">
```

```
...
```

```
<input name="calculate" type="button"  
      value="Do calculations"  
      onclick="displayResults(document.myForm) ">
```

```
...
```

```
<input name="outputAreaField" type="text"  
      readonly="true">
```

```
<input name="outputCircumferenceField"  
      type="text" readonly="true">
```

```
</form>
```

# Exemplo (3)

```
function area(r)
{ return Math.PI * r * r; }
function circumference(r)
{ return 2.0 * Math.PI * r; }

function displayResults(form)
{
  var r = parseFloat(form.radiusField.value);
  form.outputAreaField.value = area(r);
  form.outputCircumferenceField.value =
    circumference(r);
}
```

<examples/apps/circleCalculations.html>

# Gráfico de Barras

- O exemplo abaixo mostra como usar a DOM para manipular altura e largura de imagens

<examples/apps/barGraph.html>

# Um simples botão de voltar

- Como existe um objeto history, links podem ser usados para direcionar os usuários para páginas de seu histórico:

```
<a href="#"  
  onclick="history.go(-1)">Back</a>
```

- Alternativamente, um botão pode ser usado:

```
<input type="button" value="Back"  
  onclick="history.go(-1)">
```

# Mudando uma imagem (1)

- O evento **onclick** pode ser usado para substituir uma imagem por outra.
- Dê um nome para imagem **<img>**:  
****
- A imagem pode ser substituída modificando o atributo **src** usando o seguinte comando JavaScript:

```
document.myName.src = "pic2.gif";
```

# Mudando uma imagem (2)

- Documento HTML que mostra figuras contendo setas, e alguns links que mudam a cor das setas

<examples/links/linkImages1.html>

# Mudando uma imagem (3)

- As imagens precisam ser pré-carregadas:

```
var redArrow = new Image();  
redArrow.src = "redArrow.gif";  
var blueArrow = new Image();  
blueArrow.src = "blueArrow.gif";
```



# Mudando uma imagem (4)

- As duas setas são inicialmente especificadas pelas tags `<img>`

```


```

- Elas podem ser mudadas usando links:

```
<a href="#"
  onclick="changeTopToBlue()">
  Change to blue</a>
```

Também pode  
usar javascript:

# Mudando uma imagem (4)

- Funções podem ser usadas para mudar as imagens

```
function changeTopToBlue()  
{  
    window.document.topArrow.src = blueArrow.src;  
}  
function changeTopToRed()  
{  
    window.document.topArrow.src = redArrow.src;  
}  
// similar pair of functions for bottomArrow
```

# Validação de Formulários

- JavaScript pode ser usado para verificar um formulário antes dos dados serem enviados para o servidor.
- Isto reduz o número de conexões para o servidor em caso de dados incorretos ou faltantes.
- Por questões de segurança, algumas verificações precisam ser feitas do lado servidor.

# Exemplo de Validação (1)

- Escreva um formulário onde os usuários precisam informar nome e sobrenome
- Checar através de JavaScript se os dados foram informados pelo usuário

First Name:

Last Name:

Quanto o botão submit é clicado, um método javascript é chamado

# Exemplo de Validação (2)

<p>

<form name="theForm"

action="/cgi-bin/formPost.pl" method="post"

onsubmit="return checkForm(theForm.first,  
theForm.last) ">

First Name: <input type="text" name="first"><br>

Last Name: <input type="text" name="last"><br>

</p><p>

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

<input type="reset" value="reset">

</p>

</form>

Formulário será submetido apenas se  
checkForm retornar true

# Exemplo de Validação (3)

```
function checkForm(first, last)
{
    var error = "";
    if (first.value == "")
        error += "You must enter a first name";
    if (last.value == "")
        error += "\nYou must enter a last name";
    if (error == "")
        return confirm("No errors found, ...");
    else
    {
        alert("The following errors were found\n" +
            error);
        return false;
    }
}
```

# Exemplo de Validação (2)

- Código de validação

<examples/forms/formValidate.html>

# Validação de Email (1)

```
<form name="theForm"
  action="/cgi-bin/formPost.pl" method="post"
  onsubmit="return checkForm(theForm.email)">
<p>
email: <input type="text" name="first">
</p>
<p>
<input type="submit" value="submit">
<input type="reset" value="reset">
</p>
</form>
```

Formulário será submetido apenas se  
checkForm retornar true



# Validação de Email (2)

```
function checkForm(email)
{
    if (! isValidEmailAddress(email.value))
    {
        alert("Email address is invalid");
        email.focus();
        email.select();
        return false;
    }
    return true;
}
```

Foca no input

Selecionar o texto

<examples/forms/emailValidate.html>

# Trabalhando com Metadados

- Um dos usos do objeto **Document** é prover o programador com informação sobre o documento

Property	Description	Returns
characterSet	Returns the document character set encoding. This is a read-only property.	string
charset	Gets or sets the document character set encoding.	string
compatMode	Gets the compatibility mode for the document.	string
cookie	Gets or sets the cookies for the current document.	string
defaultCharset	Gets the default character encoding used by the browser.	string
defaultView	Returns the Window object for the current document; see Chapter 27 for details of this object.	Window
dir	Gets or sets the text direction for the document.	string
domain	Gets or sets the domain for the current document.	string

# Trabalhando com Metadados

- Um dos usos do objeto **Document** é prover o programador com informação sobre o documento

Property	Description	Returns
lastModified	Returns the last modified time of the document (or the current time if no modification time is available).	string
location	Provides information about the URL of the current document.	Location
readyState	Returns the state of the current document. This is a read-only property.	string
referrer	Returns the URL of the document that linked to the current document (this is the value of the corresponding HTTP header).	string
title	Gets or sets the title of the current document (the contents of the title element, described in Chapter 7).	string

# Trabalhando com Metadados

- Um dos usos do objeto **Document** é prover o programador com informação sobre o documento

```
<script>
    document.writeln("<pre>");
    document.writeln("characterSet: " + document.characterSet);
    document.writeln("charset: " + document.charset);
    document.writeln("compatMode: " + document.compatMode);
    document.writeln("Charset: " + document.defaultCharset);
    document.writeln("dir: " + document.dir);
    document.writeln("domain: " + document.domain);
    document.writeln("lastModified: " + document.lastModified);
    document.writeln("referrer: " + document.referrer);
    document.writeln("title: " + document.title);
    document.write("</pre>");
</script>
```

# Trabalhando com Metadados

- Um dos usos do objeto **Document** é prover o programador com informação sobre o documento

```
<script>
```

```
document.
```

```
document.
```

```
document.
```

```
document.
```

```
document.
```

```
document.
```

```
document.
```

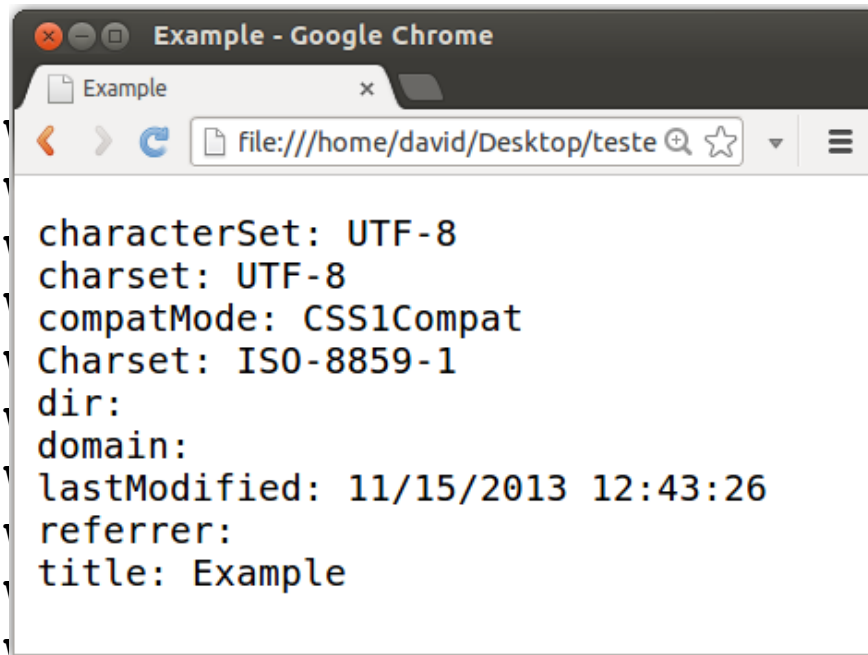
```
document.
```

```
document.
```

```
document.
```

```
document.write("</pre>");
```

```
</script>
```



```
ment.characterSet);  
charset);  
nt.compatMode);  
defaultCharset);  
;  
omain);  
ment.lastModified);  
.referrer);  
tle);
```

# Obtendo Objetos HTML

- Outra função do objeto **Document** é agir como ponte de acesso aos elementos do documento
- Propriedades do objeto Document:

Property	Description	Returns
activeElement	Returns an object representing the currently focused element	HTMLElement
body	Returns an object representing the body element	HTMLElement
embeds plugins	Returns objects representing all the embed elements	HTMLCollection
forms	Returns objects representing all the form elements	HTMLCollection
head	Returns an object representing the head element	HTMLHeadElement
images	Returns objects representing all the img elements	HTMLCollection
links	Returns objects representing all the a and area elements in	HTMLCollection

# Obtendo Objetos HTML

- Outra função do objeto **Document** é agir como ponte de acesso aos elementos do documento
- Propriedades do objeto Document:

Property	Description	Returns
activeElement	Returns an object representing the currently focused	HTML Element
body	Note que a maioria das propriedades retornam um objeto <b>HTMLCollection</b> . Este objeto representa uma coleção de elementos HTML, e pode ser acessado como um vetor.	Element
embeds		collection
plugins		
forms		HTMLCollection
head	Returns an object representing the head element	HTMLHeadElement
images	Returns objects representing all the img elements	HTMLCollection
links	Returns objects representing all the a and area elements in	HTMLCollection

# Obtendo Objetos HTML

- Exemplificando o uso de objetos HTML

```
<pre id="results"></pre>



<script>
    var resultsElement = document.getElementById("results");
    var elems = document.images;
    for (var i = 0; i < elems.length; i++) {
        resultsElement.innerHTML += "Element: " + elems[i].id + "\n";
    }
    var srcValue = elems.namedItem("apple").src;
    resultsElement.innerHTML += "SRC de apple: " + srcValue + "\n";
</script>
```



# Obtendo Objetos HTML

- Exemplificando o uso de `HTMLCollection`

```
<pre id="results"></pre>



<script>
    var resultsElement = document.getElementById("results");
    var elems = document.images;
    for (var i = 0; i < elems.length; i++) {
        resultsElement.innerHTML += "Element: " + elems[i].id + "\n";
    }
    var srcValue = elems.namedItem("apple").src;
    resultsElement.innerHTML += "SRC do apple: " + srcValue + "\n";
</script>
```

Note que usamos a propriedade **innerHTML** para setar o conteúdo de **<pre>**.

Usamos **namedItem** para selecionar o item de **HTMLCollection** que possui um determinado nome.

# Obtendo Objetos HTML

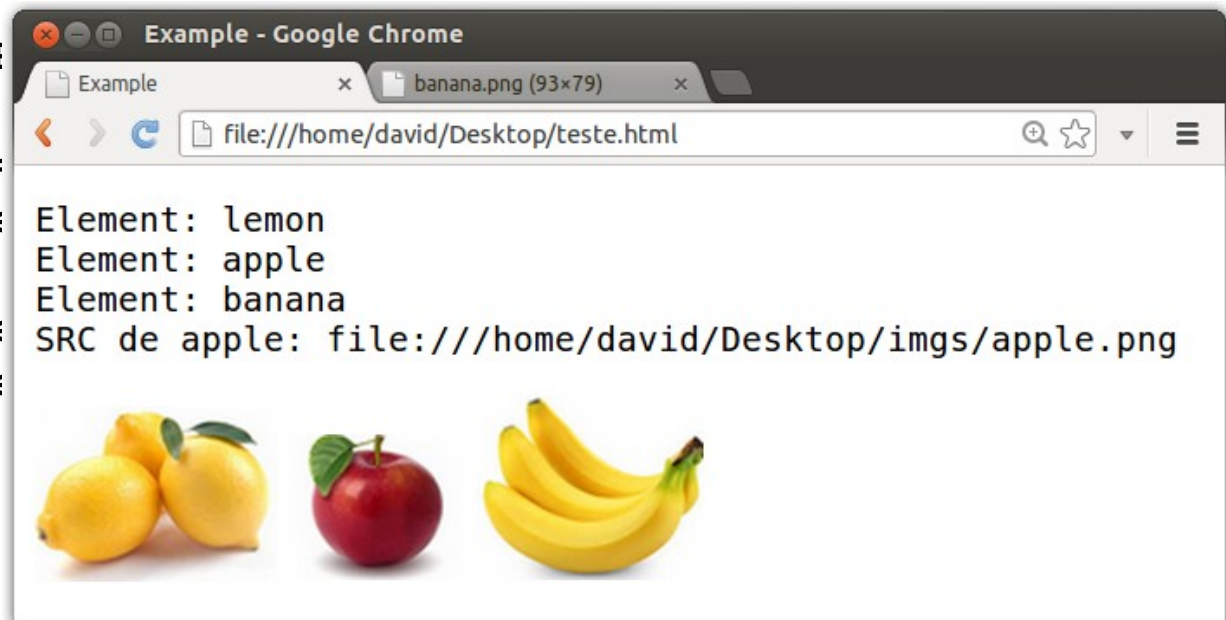
- Exemplificando o uso de objetos HTML

```
<pre id="results"></pre>



<script>
```

```
  var resultsElems = document.getElementById("results").children;
  var elems = [];
  for (var i = 0; i < resultsElems.length; i++) {
    resultsElems[i].src = "lemon.png";
  }
  var srcValue = document.getElementById("apple").src;
  resultsElems[1].src = srcValue;
</script>
```



"\n";

"\n";

# Buscando Elementos

- O objeto Document define um conjunto de métodos que podem ser usados para busca elementos no documento

Property	Description	Returns
<code>getElementById(&lt;id&gt;)</code>	Returns the element with the specified id value	HTMLElement
<code>getElementsByClassName(&lt;class&gt;)</code>	Returns the elements with the specified class value	HTMLElement[]
<code>getElementsByName(&lt;name&gt;)</code>	Returns the elements with the specified name value	HTMLElement[]
<code>getElementsByTagName(&lt;tag&gt;)</code>	Returns the elements of the specified type	HTMLElement[]
<code>querySelector(&lt;selector&gt;)</code>	Returns the first element that matches the specified CSS selector	HTMLElement
<code>querySelectorAll(&lt;selector&gt;)</code>	Returns all of the elements that match the specified CSS selector	HTMLElement[]

# Buscando Elementos

```
<pre id="results"></pre>
<p>IComp</p><p>UFAM</p>




<script>
    var resultsEl = document.getElementById("results");

    var pElems = document.getElementsByTagName("p");
    resultsEl.innerHTML += "Existem " + pElems.length
                        + " parágrafos \n";

    var fruitsElems = document.getElementsByClassName("fruits");
    resultsEl.innerHTML += "Existem " + fruitsElems.length
                        + " elementos com a classe fruits \n";

    var nameElems = document.getElementsByName("apple");
    resultsEl.innerHTML += "Existem " + nameElems.length
                        + " elementos com o nome 'apple'";
</script>
```

# Buscando Elementos

```
id="results"></pre>
<p>IComp</p><p>UFAM</p>


;

# Buscando Elementos

- Também é possível usar seletores CSS, que permitem uma seleção mais versátil dos elementos
- No exemplo abaixo, usamos um seletor para selecionar todos os elementos **p** e a imagem com **id=apple**

```
<script>
  var resultsElement = document.getElementById("results");
  var elems = document.querySelectorAll("p, img#apple")
  resultsElement.innerHTML += "O seletor encontrou "
                           + elems.length + " elementos\n";
</script>
```

# Trabalhando com Window

- O objeto **Window** foi adicionado ao HTML como parte da especificação do HTML5
- Antes do HTML5, este objeto existia como um padrão não oficial, suportado pela maioria dos browsers
- Conforme o nome sugere, a funcionalidade básica de windows está relacionada com a janela onde documento está sendo apresentado
- Este objeto pode ser acessado diretamente através da variável global **window**

# Trabalhando com Window

- Propriedades de `window`

| Name                     | Description                                                                                       | Returns |
|--------------------------|---------------------------------------------------------------------------------------------------|---------|
| <code>innerHeight</code> | Gets the height of the window content area                                                        | number  |
| <code>innerWidth</code>  | Gets the width of the window content area                                                         | number  |
| <code>outerHeight</code> | Gets the height of the window, including borders, menu bars, and so on                            | number  |
| <code>outerWidth</code>  | Gets the width of the window, including borders, menu bars, and so on                             | number  |
| <code>pageXOffset</code> | Gets the number of pixels that the window has been scrolled horizontally from the top-left corner | number  |
| <code>pageYOffset</code> | Gets the number of pixels that the window has been scrolled vertically from the top-left corner   | number  |
| <code>screen</code>      | Returns a Screen object describing the screen                                                     | Screen  |



# Trabalhando com Window

- O objeto Window também provê um conjunto de métodos de interação com a janela que contém o documento:

| Name                                        | Description                                                             | Returns |
|---------------------------------------------|-------------------------------------------------------------------------|---------|
| <code>blur()</code>                         | Unfocuses the window                                                    | void    |
| <code>close()</code>                        | Closes the window (not all browsers allow a script to close the window) | void    |
| <code>focus()</code>                        | Focuses the window                                                      | void    |
| <code>print()</code>                        | Prompts the user to print the page                                      | void    |
| <code>scrollBy(&lt;x&gt;, &lt;y&gt;)</code> | Scrolls the document relative to its current position                   | void    |
| <code>scrollTo(&lt;x&gt;, &lt;y&gt;)</code> | Scrolls to the specified position                                       | void    |
| <code>stop()</code>                         | Stops the document from loading                                         | void    |

# Trabalhando com Classes

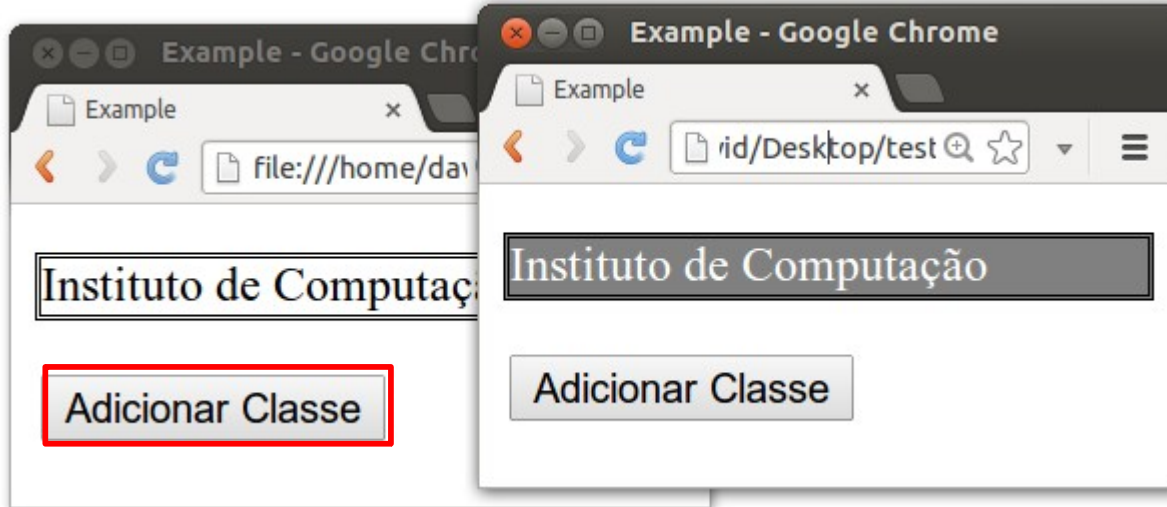
- Através da propriedade `className`, pode-se recuperar ou editar a lista das classes de um elemento
  - Pode-se remover ou adicionar classes apenas mudando o valor dessa propriedade

```
<style>
p { border: medium double black; }
p.newclass { background-color: grey; color: white; }
</style>
...
<p id="textblock">Instituto de Computação</p>
<button id="pressme">Adicionar Classe</button>
<script>
document.getElementById("pressme").onclick = function(e) {
    document.getElementById("textblock").className += " newclass";
};
</script>
```

# Trabalhando com Classes

- Através da propriedade `className`, pode-se recuperar ou editar a lista das classes de um elemento
  - Pode-se remover ou adicionar classes apenas mudando o valor dessa propriedade

```
<style>
p { border
p.newclass
</style>
...
<p id="tex
<button id
<script>
document.g
document
};
</script>
```



```
e) {
newclass";
```

# Trabalhando com Atributos

- Também é possível ler e editar qualquer atributo de um dado elemento

Member	Description	Returns
attributes	Returns the attributes applied to the element	Attr[ ]
dataset	Returns the data-* attributes	string[<name>]
getAttribute(<name>)	Returns the value of the specified attribute	string
hasAttribute(<name>)	Returns true if the element has the specified attribute	boolean
removeAttribute(<name>)	Removes the specified attribute from the element	void
setAttribute(<name>, <value>)	Applies an attribute with the specified name and value	void

# Trabalhando com Atributos

- Também é possível ler e editar qualquer atributo de um dado elemento

```
<p id="textblock">Instituto de Computação</p>
<pre id="results"></pre>
<script>
    var results = document.getElementById("results");
    var elem = document.getElementById("textblock");
    results.innerHTML = "Element has lang attribute: "
                      + elem.hasAttribute("lang") + "\n";

    results.innerHTML += "Adding lang attribute\n";
    elem.setAttribute("lang", "en-US");
    results.innerHTML += "Attr value is : "
                      + elem.getAttribute("lang") + "\n";
    results.innerHTML += "Set new value for lang attribute\n";
    elem.setAttribute("lang", "en-UK");
    results.innerHTML += "Value is now: "
                      + elem.getAttribute("lang") + "\n";
</script>
```

# Trabalhando com Atributos

- Também é possível ler e editar qualquer atributo de um dado elemento

```
<p id="textblock">
<pre id="result">
<script>
```

```
var results =
var elem = doc
results.inner
```

```
results.inner
elem.setAttribute
results.inner
```

```
results.inner
```

```
elem.setAttribute("lang", "en-UK");
```

```
results.innerHTML += "Value is now: "
```

```
+ elem.getAttribute("lang") + "\n";
```

```
</script>
```



# Trabalhando com Atributos

- É possível acessar todos os atributos de um elemento através da propriedade **attributes**, que retorna um vetor de objetos **Attr**
- Propriedades de um objeto **Attr**:

Properties	Description	Returns
name	Returns the name of the attribute	string
value	Gets or sets the value of the attribute	string

# Trabalhando com Atributos

- Usando `attributes` e `Attr`

```
<p id="textblock" data-nome="icomp" data-sentimento="like">
  Instituto de Computação
</p>
<pre id="results"></pre>
```

```
<script>
  var results = document.getElementById("results");
  var elem = document.getElementById("textblock");
  var attrs = elem.attributes;
  for (var i = 0; i < attrs.length; i++) {
    results.innerHTML += "Name: " + attrs[i].name +
      " Value: " + attrs[i].value + "\n";
  }
  attrs["data-nome"].value = "icomp/ufam";
  results.innerHTML += "Value of data-nome attr: "
    + attrs["data-nome"].value;
</script>
```

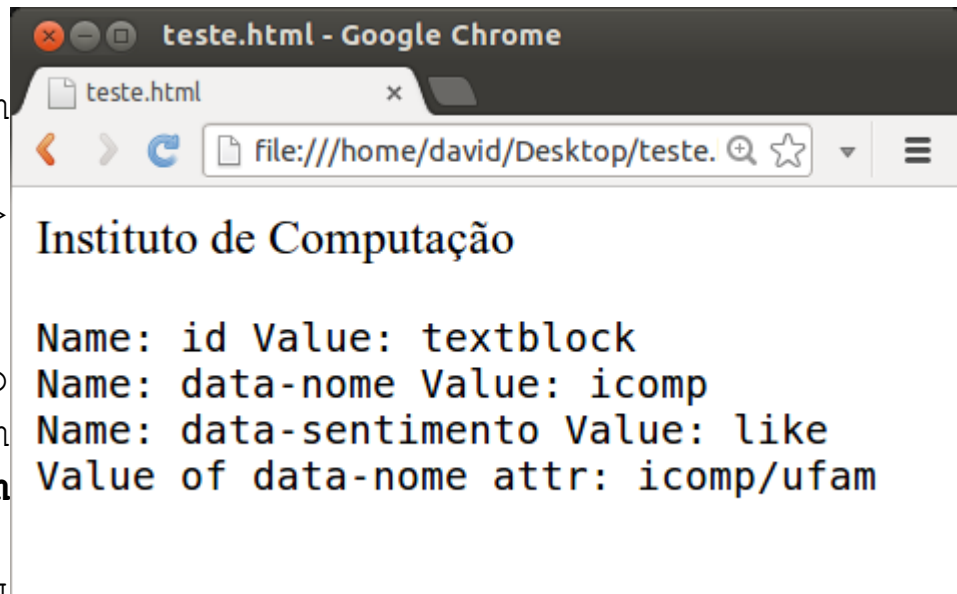


# Trabalhando com Atributos

- Usando attributes e Attr

```
<p id="textblock"
  Instituto de Com
</p>
<pre id="results">

<script>
  var results = do
  var elem = docum
  var attrs = elem
  for (var i = 0;
    results.innerHTML += "Name: " + attrs[i].name + " Value: " + attrs[i].value + "\n";
  }
  attrs["data-nome"].value = "icomp/ufam";
  results.innerHTML += "Value of data-nome attr: "
    + attrs["data-nome"].value;
</script>
```



# Modificando o modelo

- Nos últimos slides, foi apresentado como usar a DOM para manipular elementos individuais
  - Foi mostrado, por exemplo, como mudar os atributos e o conteúdo textual dos elementos
- Isso é possível porque existe um link em tempo real entre a DOM e o documento HTML
- É possível usar esse link para irmos além das manipulações até então apresentadas, por exemplo mudando a estrutura do documento HTML

# Modificando o modelo

- Funções de manipulação da árvore DOM

Member	Description	Returns
<code>appendChild(HTMLElement)</code>	Appends the specified element as a child of the current element	HTMLElement
<code>cloneNode(boolean)</code>	Copies an element	HTMLElement
<code>compareDocumentPosition(HTMLElement)</code>	Determines the relative position of an element	number
<code>innerHTML</code>	Gets or sets the element's contents	string
<code>insertAdjacentHTML(&lt;pos&gt;, &lt;text&gt;)</code>	Inserts HTML relative to the element	void
<code>insertBefore(&lt;newElem&gt;, &lt;childElem&gt;)</code>	Inserts the first element before the second (child) element	HTMLElement

# Modificando o modelo

- Funções de manipulação da árvore DOM

Member	Description	Returns
<code>isSameNode(HTMLElement)</code>	Determines if the specified element is the same as the current element	boolean
<code>outerHTML</code>	Gets or sets an element's HTML and contents	string
<code>removeChild(HTMLElement)</code>	Removes the specified child of the current element	HTMLElement
<code>replaceChild(HTMLElement, HTMLElement)</code>	Replaces a child of the current element	HTMLElement

# Modificando o modelo

- Essas propriedades e métodos estão disponíveis para todos os objetos de elementos
- Adicionalmente, o objeto **Document** define dois métodos para criação de novos elementos
  - Esses métodos são essenciais para adicionar conteúdo ao documento HTML

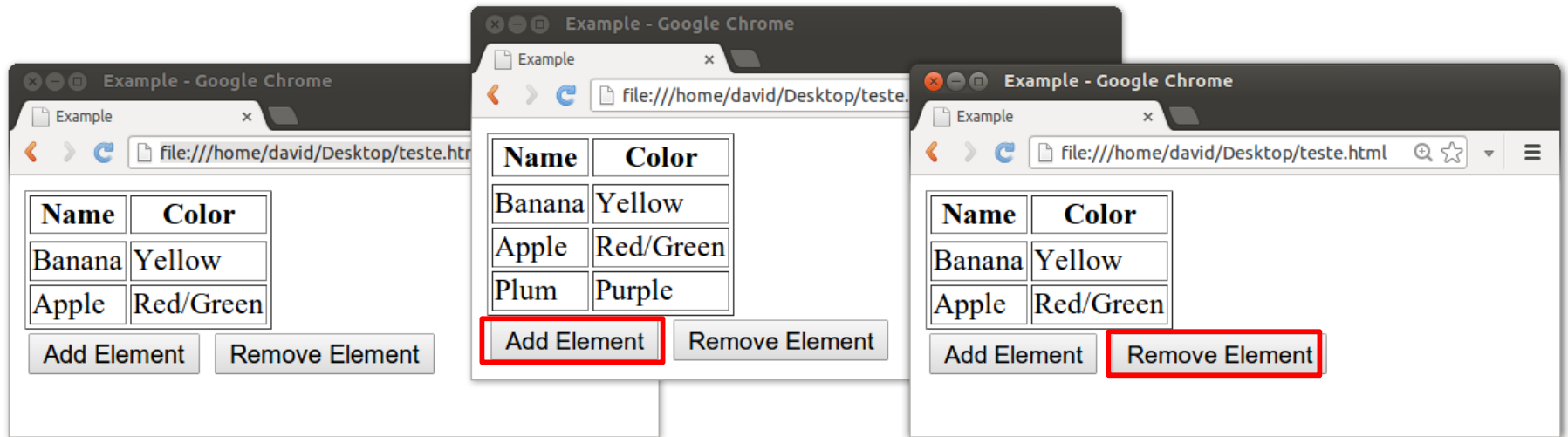
Member	Description	Returns
<code>createElement(&lt;tag&gt;)</code>	Creates a new <code>HTMLElement</code> object with the specific tag type	<code>HTMLElement</code>
<code>createTextNode(&lt;text&gt;)</code>	Creates a new <code>Text</code> object with the specified content	<code>Text</code>

# Criando e Apagando Elementos

```
<table border="1">
  <thead><th>Name</th><th>Color</th></thead>
  <tbody id="fruitsBody">
    <tr><td>Banana</td><td>Yellow</td></tr>
    <tr><td>Apple</td><td>Red/Green</td></tr>
  </tbody>
</table>
<button id="add">Add Element</button>
<button id="remove">Remove Element</button>
<script>
  var tableBody = document.getElementById("fruitsBody");
  document.getElementById("add").onclick = function() {
    var row = tableBody.appendChild(document.createElement("tr"));
    row.setAttribute("id", "newrow");
    row.appendChild(document.createElement("td"))
      .appendChild(document.createTextNode("Plum"));
    row.appendChild(document.createElement("td"))
      .appendChild(document.createTextNode("Purple"));
  };
  document.getElementById("remove").onclick = function() {
    var row = document.getElementById("newrow");
    row.parentNode.removeChild(row);
  }
</script>
```

# Criando e Apagando Elementos

```
<table border="1">
  <thead><th>Name</th><th>Color</th></thead>
  <tbody id="fruitsBody">
    <tr><td>Banana</td><td>Yellow</td></tr>
    <tr><td>Apple</td><td>Red/Green</td></tr>
  </tbody>
```



```
    row.appendChild(document.createElement("td"))
        .appendChild(document.createTextNode("Purple"));
};
document.getElementById("remove").onclick = function() {
    var row = document.getElementById("newrow");
    row.parentNode.removeChild(row);
}
</script>
```

# Manipulando a Estrutura

- Também é possível usar as propriedades **outerHTML** e **innerHTML** para manipular a estrutura dos documentos
  - Uma vantagem de dessas funções é que elas não requerem a criação manual de novos elementos DOM



# Manipulando a Estrutura

```
<table border="1">
  <thead><tr><th>Fruit</th><th>Color</th></tr></thead>
  <tbody><tr><td>Banana</td><td>Yellow</td></tr>
    <tr id="apple"><td>Apple</td><td>Red/Green</td></tr>
  </tbody>
</table>
<table border="1">
  <thead><tr><th>Fruit</th><th>Color</th></tr></thead>
  <tbody id="fruitsBody">
    <tr><td>Plum</td><td>Purple</td></tr>
    <tr id="targetrow"><td colspan="2">This is the placeholder</td></tr>
  </tbody>
</table>
<button id="move">Move Row</button>
<script>
  document.getElementById("move").onclick = function() {
    var source = document.getElementById("apple");
    var target = document.getElementById("targetrow");
    target.innerHTML = source.innerHTML;
    source.outerHTML = '<tr id="targetrow"><td colspan="2">' +
      'This is the placeholder</td>';
  };
</script>
```

# Manipulando a Estrutura

```
<table border="1">
  <thead><tr><th>Fruit</th><th>Color</th></tr>
  <tbody><tr><td>Banana</td><td>Yellow</td></tr>
  <tr id="apple"><td>Apple</td><td>Red/Green</td></tr>
</tbody>
</table>
```

```
<table border="1">
  <thead><tr><th>Fruit</th><th>Color</th></tr>
  <tbody id="fruitsBody">
    <tr><td>Plum</td><td>Purple</td></tr>
    <tr id="targetrow"><td colspan="2">This is the placeholder</td></tr>
  </tbody>
</table>
```

```
<button id="move">Move Row</button>
<script>
```

```
  document.getElementById("move").onclick = function() {
```

```
    var source = document.getElementById("apple");
```

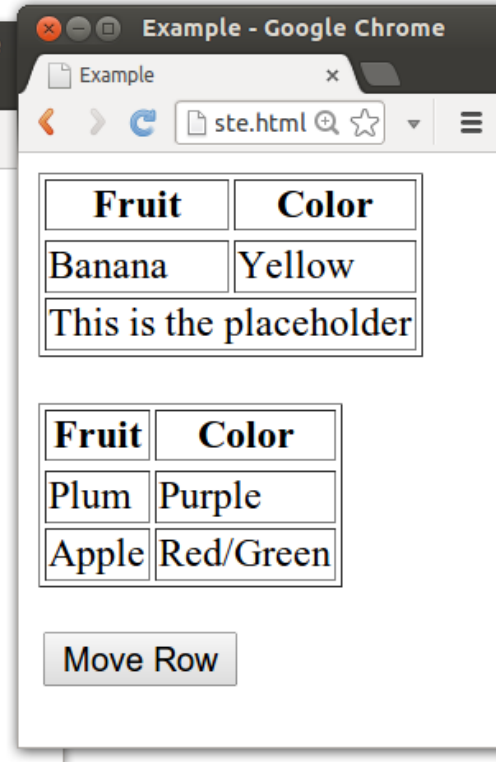
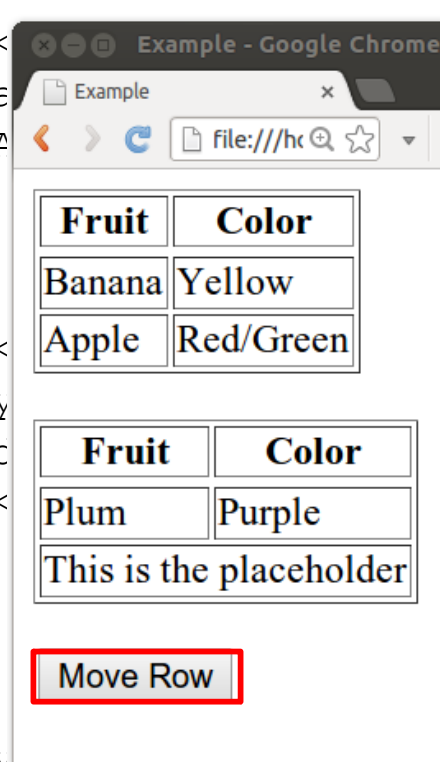
```
    var target = document.getElementById("targetrow");
```

```
    target.innerHTML = source.innerHTML;
```

```
    source.outerHTML = '<tr id="targetrow"><td colspan="2">' +
      'This is the placeholder</td></tr>';
```

```
  };
```

```
</script>
```



r</td></tr>

# Trabalhando com Stylesheets

- Para obter os estilos de um determinado elemento, usamos a propriedade **style**
  - Essa propriedade é definida para todo objeto **HTMLElement**
- A propriedade **style** retorna um objeto **CSSStyleDeclaration**, que por sua vez contém uma propriedade chamada **cssText**
  - Essa propriedade pode ser usada para ler e/ou modificar os estilos aplicados a um dado elemento

```
var icomp = document.getElementById("icomp");  
document.getElementById("pressme").onclick = function() {  
    icomp.style.cssText = "color:black";  
}
```

# Objetos **CSSStyleDeclaration**

- Para obter total controle do CSS através da DOM, precisamos usar o objeto **CSSStyleDeclaration**
- Membros do objeto **CSSStyleDeclaration**:

Member	Description	Returns
cssText	Gets or sets the text of the style.	string
getPropertyCSSValue(<name>)	Gets the specified property.	CSSPrimitiveValue
getPropertyPriority(<name>)	Gets the priority of the specified property.	string
getPropertyValue(<name>)	Gets the specified value as a string.	string
removeProperty(<name>)	Removes the specified property.	string
setProperty(<name>, <value>, <priority>)	Sets the value and priority for the specified property.	void
<style>	Convenience property to get or set the specified CSS property.	string

# Objetos **CSSStyleDeclaration**

- A forma mais fácil de usar um objeto **CSSStyleDeclaration** é através do acesso direto as propriedades CSS

```
<body>
  <div id="icomp">Instituto de Computação</div>
  <button id="pressme">Mudar Estilo</button>

  <script>
    var icomp = document.getElementById("icomp");
    document.getElementById("pressme").onclick = function() {
      icomp.style.fontSize = "32px";
      icomp.style.color = "red";
    }
  </script>

</body>
```

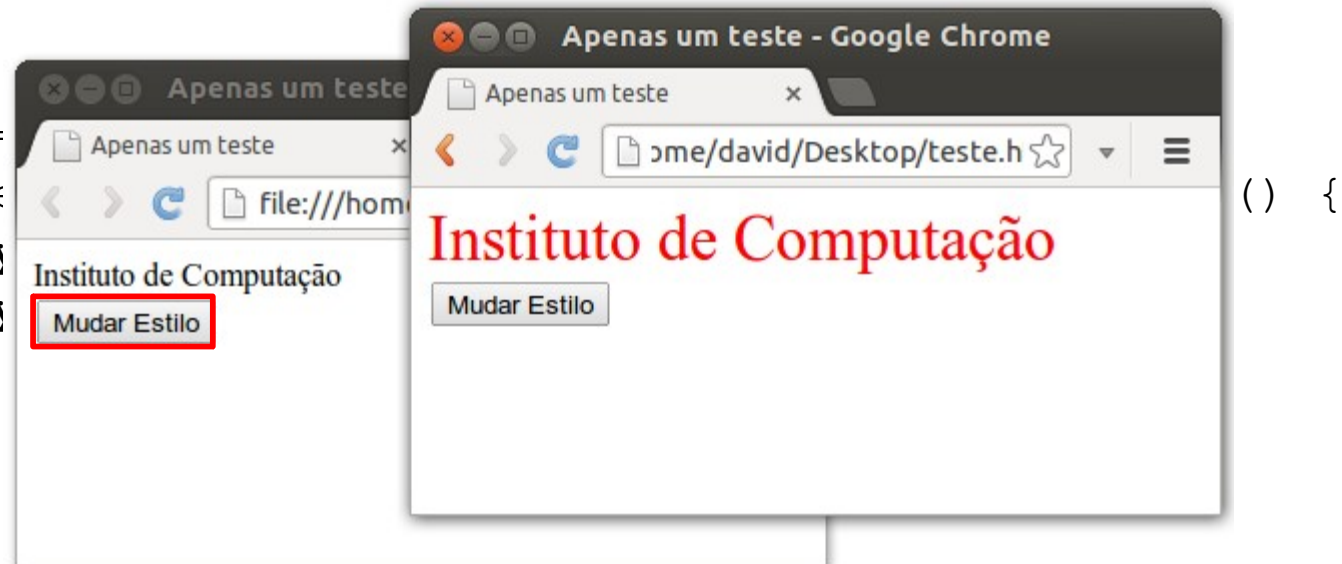
# Objetos CSSStyleDeclaration

- A forma mais fácil de usar um objeto **CSSStyleDeclaration** é através do acesso direto as propriedades CSS

```
<body>  
  <div id="icomp">Instituto de Computação</div>  
  <button id="pressme">Mudar Estilo</button>
```

```
<script>  
  var icomp =  
  document.ge  
    icomp.sty  
    icomp.sty  
  }  
</script>
```

```
</body>
```



# Objetos **CSSStyleDeclaration**

- Também podemos usar as propriedades **setProperty** e **removeProperty** para manipular os estilos

```
<div id="icomp">Instituto de Computação</div>
<button id="add">Adicionar Estilos</button>
<button id="clean">Limpar Estilo</button>
<script>
  var icomp = document.getElementById("icomp");
  document.getElementById("add").onclick = function() {
    icomp.style.setProperty("background-color", "lightgray");
    icomp.style.setProperty("color", "red");
    icomp.style.setProperty("font-size", "32px");
  }
  document.getElementById("clean").onclick = function() {
    icomp.style.removeProperty("background-color");
    icomp.style.removeProperty("color");
    icomp.style.removeProperty("font-size");
  }
</script>
```





# Objetos CSSStyleDeclaration

- Também é possível acessar a lista de estilos dos elementos

```
<div id="icomp">Instituto de Computação</div>
<button id="add">Adicionar Estilos</button>
<button id="clean">Limpar Estilo</button>
<div id="estilos"></div>
```

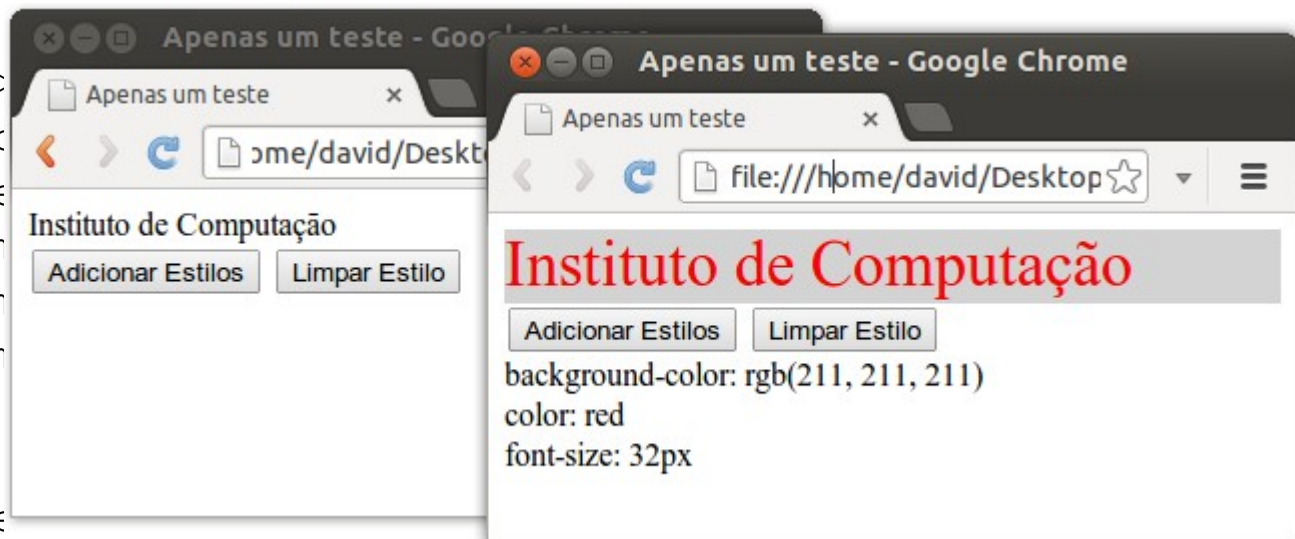
```
<script>
  var icomp = document.getElementById("icomp");
  document.getElementById("add").onclick = function() {
    icomp.style.setProperty("background-color", "lightgray");
    icomp.style.setProperty("color", "red");
    icomp.style.setProperty("font-size", "32px");
    var estilos = document.getElementById("estilos");
    for (var i = 0; i < icomp.style.length; i++) {
      estilos.innerHTML += icomp.style[i] + ": " +
        icomp.style.getPropertyValue(icomp.style[i]) +
        "<br />" ;
    }
  }
</script>
```

# Objetos CSSStyleDeclaration

- Também é possível acessar a lista de estilos dos elementos

```
<div id="icomp">Instituto de Computação</div>
<button id="add">Adicionar Estilos</button>
<button id="clean">Limpar Estilo</button>
<div id="estilos"></div>
```

```
<script>
var icomp = document.getElementById("icomp");
var estilos = document.getElementById("estilos");
for (var i = 0; i < estilos.length; i++) {
```



```
icomp.style.getPropertyValue(icomp.style[i]) +
"<br />" ;
```

```
    }
  }
</script>
```

# Trabalhando com Eventos

- A forma mais simples de usar eventos é adicionar um atributo HTML em determinado elemento HTML, de acordo com o tipo de evento desejado
  - O valor do atributo deve ser um conjunto de comandos JavaScript

```
<p onmouseover="this.style.background='black';  
this.style.color='white'">
```

O Instituto de Computação (IComp) é o mais novo instituto da UFAM, tendo sido formado a partir do antigo Departamento de Ciência da Computação (DCC). Este instituto agrega os professores da área de computação. Como toda unidade acadêmica, o IComp atua no ensino, pesquisa e extensão, além de desempenhar atividades administrativas.

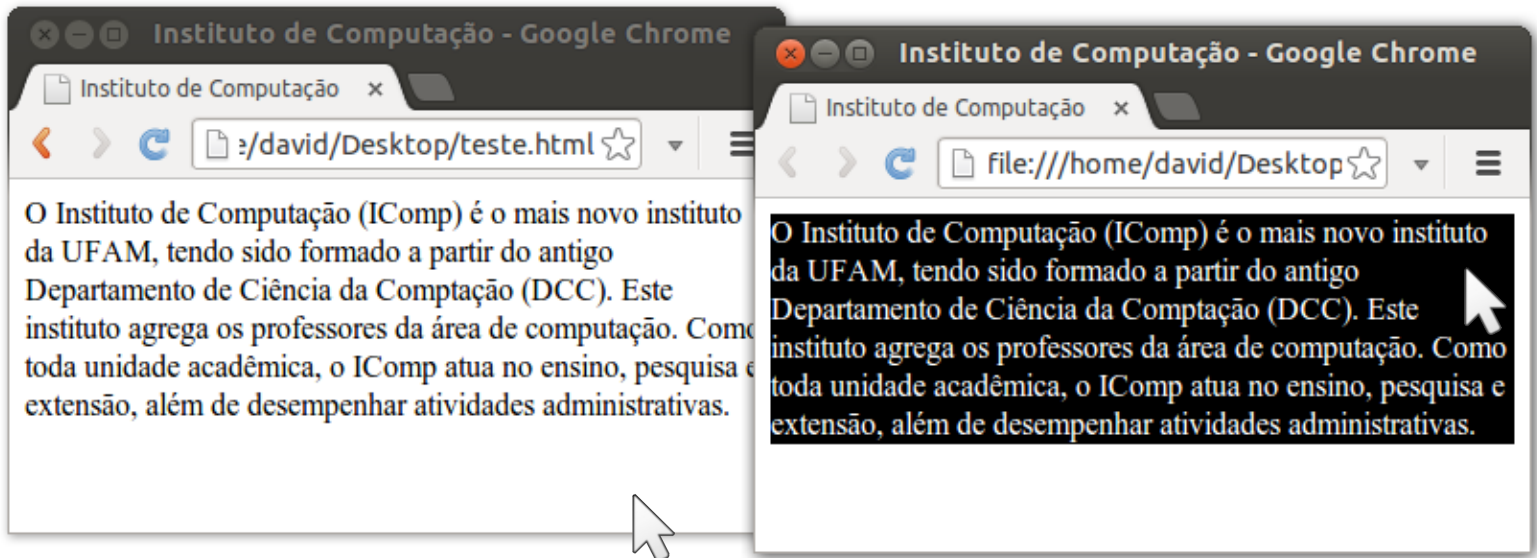
```
</p>
```

# Trabalhando com Eventos

- A forma mais simples de usar eventos é adicionar um atributo HTML em determinado elemento HTML, de acordo com o tipo de evento desejado
  - O valor do atributo deve ser um conjunto de comandos JavaScript

```
<p onmouseover="this.style.background='black';  
this.style.color='white'">
```

O Ir  
da U  
de C  
prof  
acad  
de c  
</p>



# Trabalhando com Eventos

- A forma mais simples de usar eventos é adicionar um atributo HTML em determinado elemento HTML, de acordo com o tipo de evento desejado
  - O valor do atributo deve ser um conjunto de comandos JavaScript

```
<p onmouseover="this.style.background='white';  
this.style.color='black' "  
onmouseout="this.style.removeProperty('color');  
this.style.removeProperty('background') ">
```

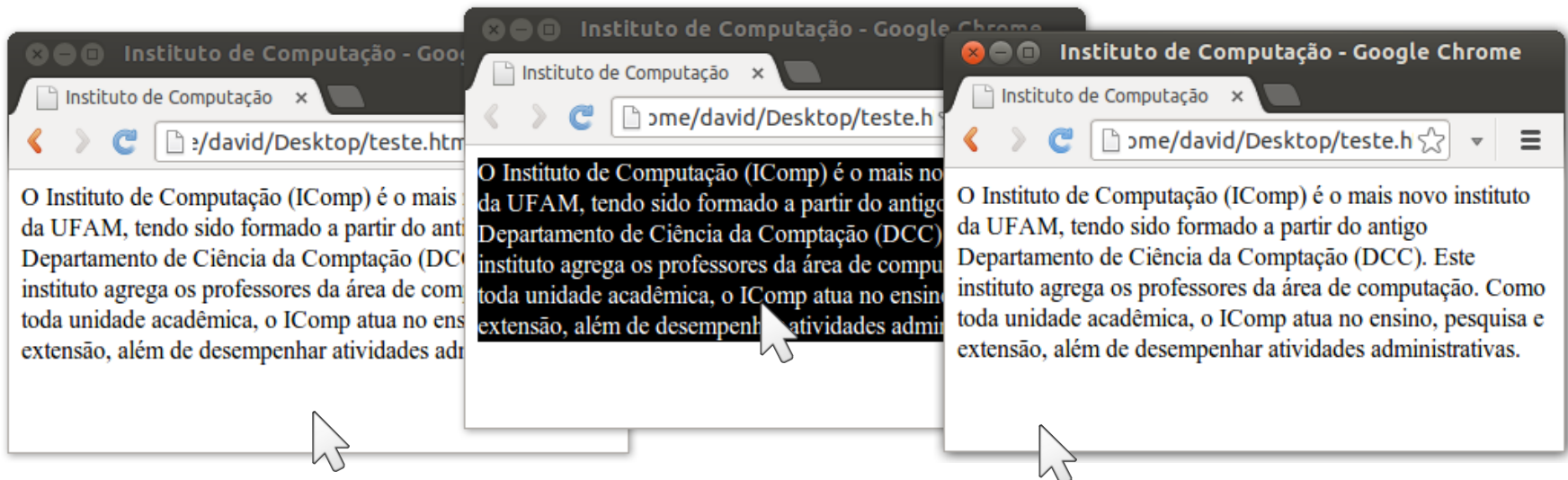
O Instituto de Computação (IComp) é o mais novo instituto da UFAM, tendo sido formado a partir do antigo Departamento de Ciência da Computação (DCC). Este instituto agrega os professores da área de computação. Como toda unidade acadêmica, o IComp atua no ensino, pesquisa e extensão, além de desempenhar atividades administrativas.

```
</p>
```

# Trabalhando com Eventos

- A forma mais simples de usar eventos é adicionar um atributo HTML em determinado elemento HTML, de acordo com o tipo de evento desejado
  - O valor do atributo deve ser um conjunto de comandos JavaScript

```
<p onmouseover="this.style.background='white';  
this.style.color='black';"  
onmouseout="this.style.removeProperty('color');  
this.style.removeProperty('background')">
```



# Trabalhando com Eventos

- Também é possível chamar funções em resposta aos eventos

```
<p onmouseover="handleMouseOver(this) "
onmouseout="handleMouseOut(this) ">
```

O Instituto de Computação (IComp) é o mais novo instituto da UFAM, tendo sido formado a partir do antigo Departamento de Ciência da Computação (DCC). Este instituto agrega os professores da área de computação.

```
</p>
```

```
<script type="text/javascript">
function handleMouseOver(elem) {
    elem.style.background='black';
    elem.style.color='white';
}
function handleMouseOut(elem) {
    elem.style.removeProperty('color');
    elem.style.removeProperty('background');
}
</script>
```

# Trabalhando com Eventos

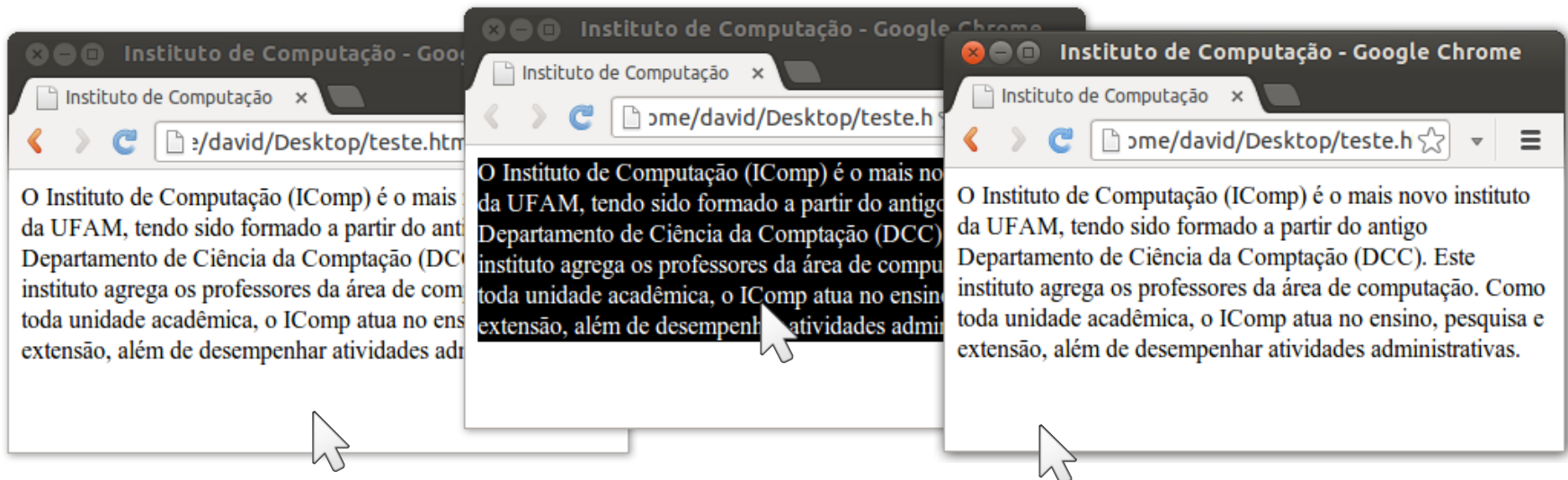
- Também é possível chamar funções em resposta aos eventos

```
<p onmouseover="handleMouseOver(this) "  
onmouseout="handleMouseOut(this) ">
```

O Instituto de Computação (IComp) é o mais novo instituto da UFAM, tendo sido formado a partir do antigo Departamento de Ciência da Computação (DCC). Este instituto agrega os professores da área de computação.

```
</p>
```

```
<script type="text/javascript">
```





# Usando o objeto Event

- Para tratar os eventos de forma mais sofisticada, precisamos trabalhar com o objeto javascript **Event**

```
<script type="text/javascript">
  var pElems = document.getElementsByTagName("p");
  for (var i = 0; i < pElems.length; i++) {
    pElems[i].onmouseover = handleMouseOver;
    pElems[i].onmouseout = handleMouseOut;
  }
```

```
function handleMouseOver(e) {
  e.target.style.background='black';
  e.target.style.color='white';
}
```

Objeto Event

```
function handleMouseOut(e) {
  e.target.style.removeProperty('color');
  e.target.style.removeProperty('background');
}
```

```
</script>
```

# Usando o objeto Event

- Para tratar os eventos de forma mais sofisticada, precisamos trabalhar com o objeto javascript **Event**

```
<script type="text/javascript">
  var pElems = document.getElementsByTagName("p");
  for (var i = 0; i < pElems.length; i++) {
    pElems[i].onmouseover = handleMouseOver;
    pElems[i].onmouseout = handleMouseOut;
  }
```

```
function handleMouseOver(e) {
  e.target.style.background='black';
  e.target.style.color='white';
}
```

Objeto Event

```
function handleMouseOut(e) {
  e.target.style.removeProperty('color');
  e.target.style.removeProperty('background');
}
```

```
</script>
```





# Acessando a DOM (1)

- O método `getElementById` pode ser usado para acessar qualquer elemento:
- O elemento

```
<div id="myTitle"...>Hello there</div>
```

pode ser acessado usando

```
var obj =
```

```
document.getElementById("myTitle");
```

# Acessando a DOM (2)

- Cada propriedade CSS possui uma propriedade correspondente na árvore DOM
- Para mudar a cor de um elemento, basta usar

```
obj.style.color = "yellow";
```

- Para mudar a cor do background usa-se

```
obj.style.backgroundColor = "red";
```

<examples/dom/colorChange.html>

# Acessando a DOM (3)

- Modificando a propriedade display do CSS para tornar visível blocos de texto.
- A propriedade display pode ser block ou none

<examples/dom/toggleText.html>