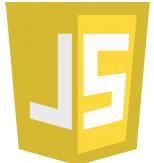


JavaScript

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

As 3 camadas

- Conforme já visto, o desenvolvimento client-side é baseado em 3 camadas principais:
 - **Conteúdo**, viabilizado pelo **HTML**
 - **Estilo**, viabilizado pelo **CSS**
 - **Comportamento**, viabilizado pelo **JavaScript**
- As camadas possibilitam o desenvolvimento independente de cada área da produção
 - Se quisermos modificar o design, podemos fazê-lo manipulando apenas o CSS, sem se preocupar com HTML ou Javascript
- Nesta primeira parte do curso, iremos abordar a terceira camada: **o Comportamento**



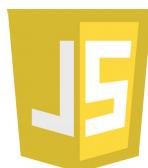
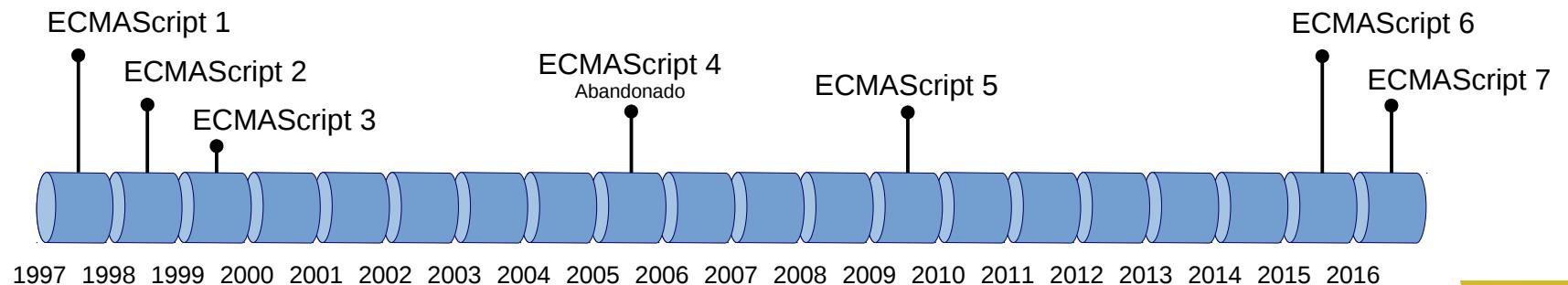
A Linguagem JavaScript

- JavaScript é uma linguagem criada por Brendan Eich durante sua passagem pela Netscape em 1995
- Não possui conexão com Java, a não ser pela sintaxe e pelo nome
 - O nome JavaScript surgiu de uma estratégia de marketing de pegar carona na popularidade da linguagem Java
- É uma linguagem **interpretada**, de **alto nível**, possui **tipagem dinâmica**
- Aceita os paradigmas **funcional** e **orientação a objetos**
- Atualmente é a principal linguagem para **programação client-side** em browsers
- Possui um nome oficial: ECMAScript



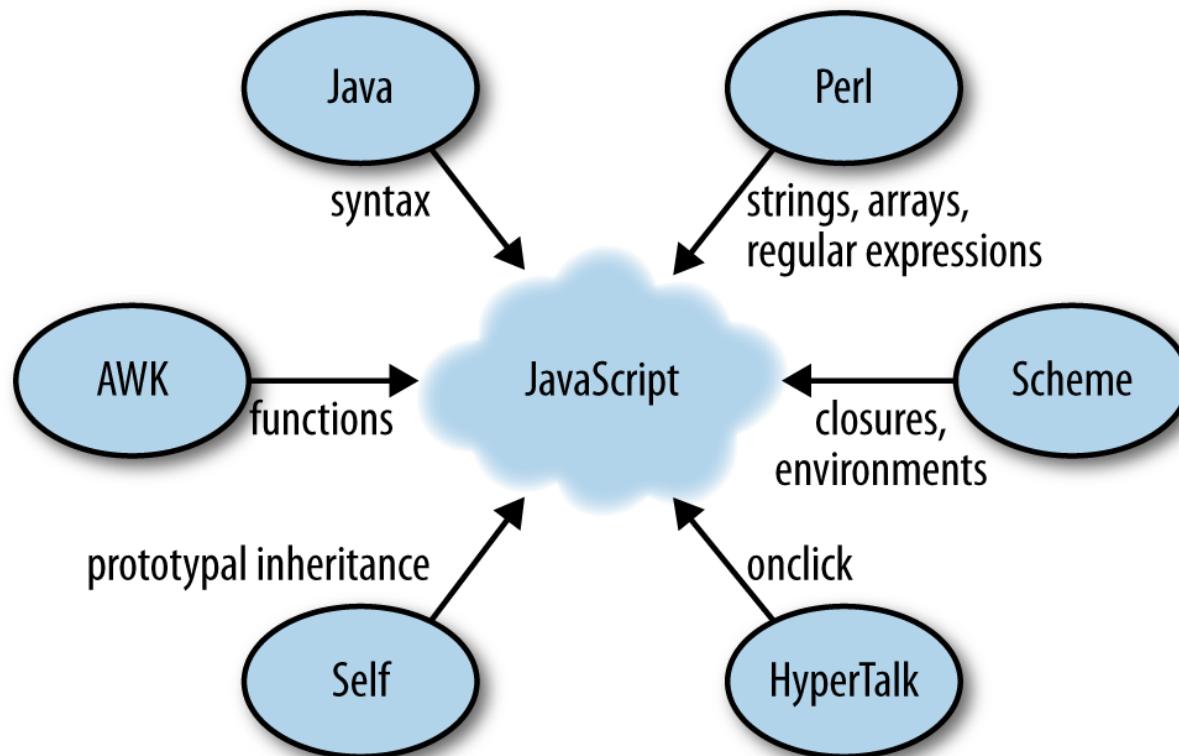
A Linguagem JavaScript

- O ECMAScript (ES) é a linguagem de scripts padronizada pelo ECMA-262
- O JavaScript é compatível com o ECMAScript, porém provê recursos adicionais não descritos na especificação ECMA
- Breve histórico do ECMAScript:



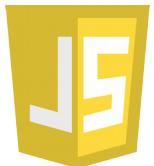
A Linguagem JavaScript

- A linguagem ECMAScript, ou JavaScript, sofreu influência de diversas linguagens de programação:



A Linguagem JavaScript

- Uma das linguagens **mais populares da atualidade**
 - Grande comunidade, várias conferências
 - Deu origem a vários frameworks e bibliotecas
 - Uso em televisores Smart TV
 - Uso em sistemas operacionais na forma de widgets
 - Plugins para softwares gráficos como Photoshop
 - Plugins para browsers
- No entanto, também é uma das **linguagens mais odiadas!**



FrameWorks JavaScript



jQuery
write less, do more.



UNDERScore.JS



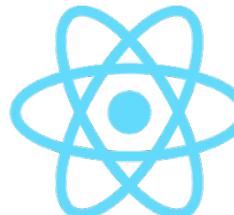
tipJS



dojo

three.js

ECHO.JS



ember.js

mootools
A COMPACT JAVASCRIPT FRAMEWORK

BACKBONE.JS
CSS



by Telerik

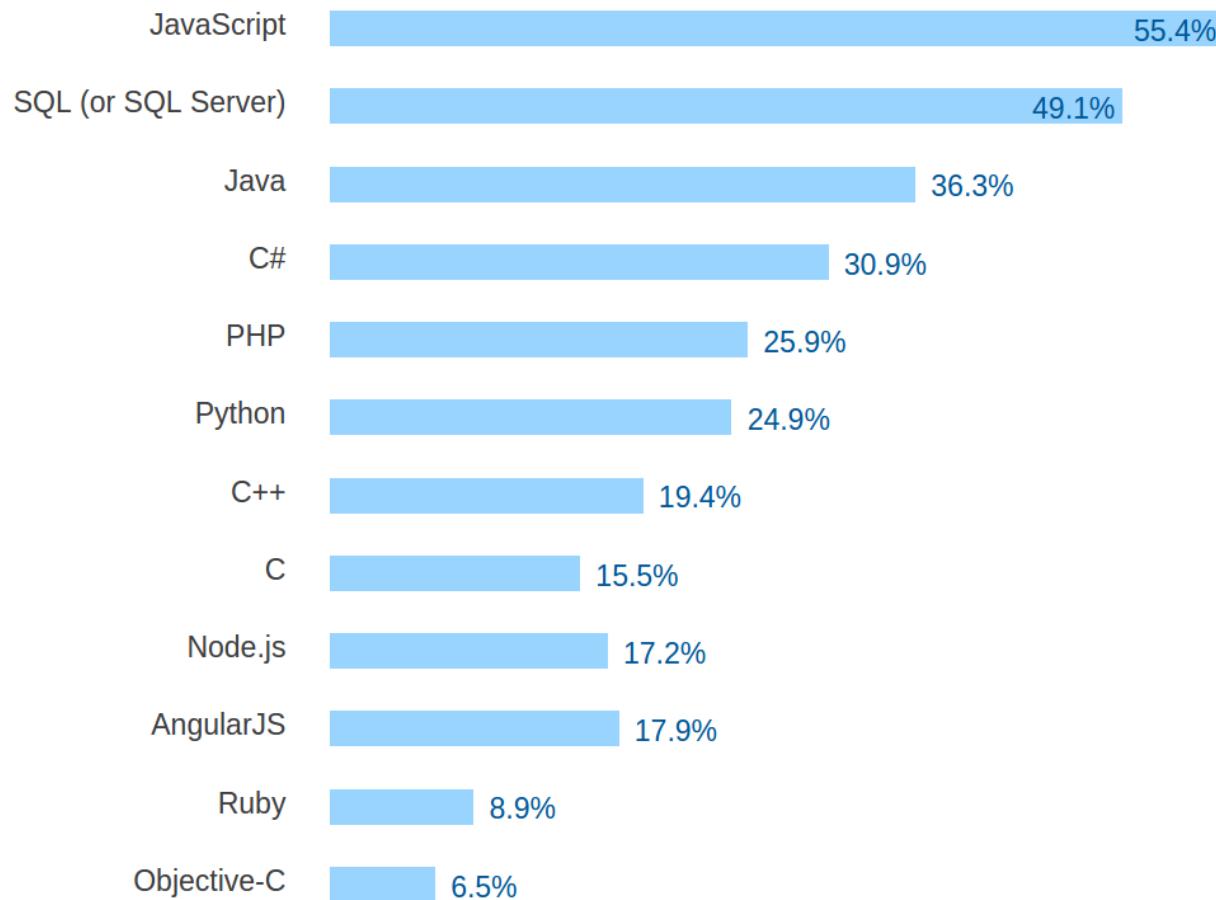
Postagens no StackOverflow

2016

2015

2014

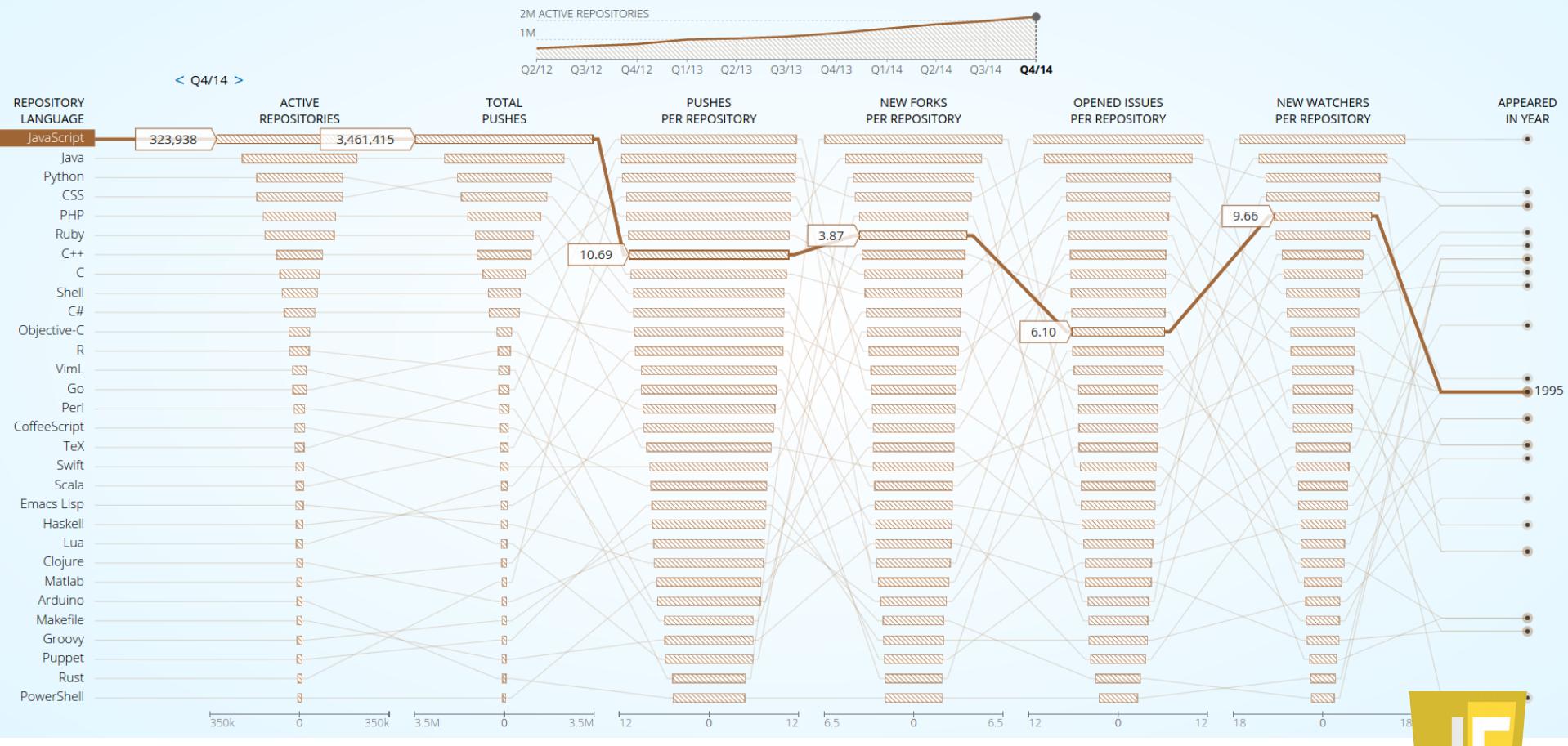
2013



Popularidade no GitHub

GitHut

A SMALL PLACE TO DISCOVER LANGUAGES IN GITHUB



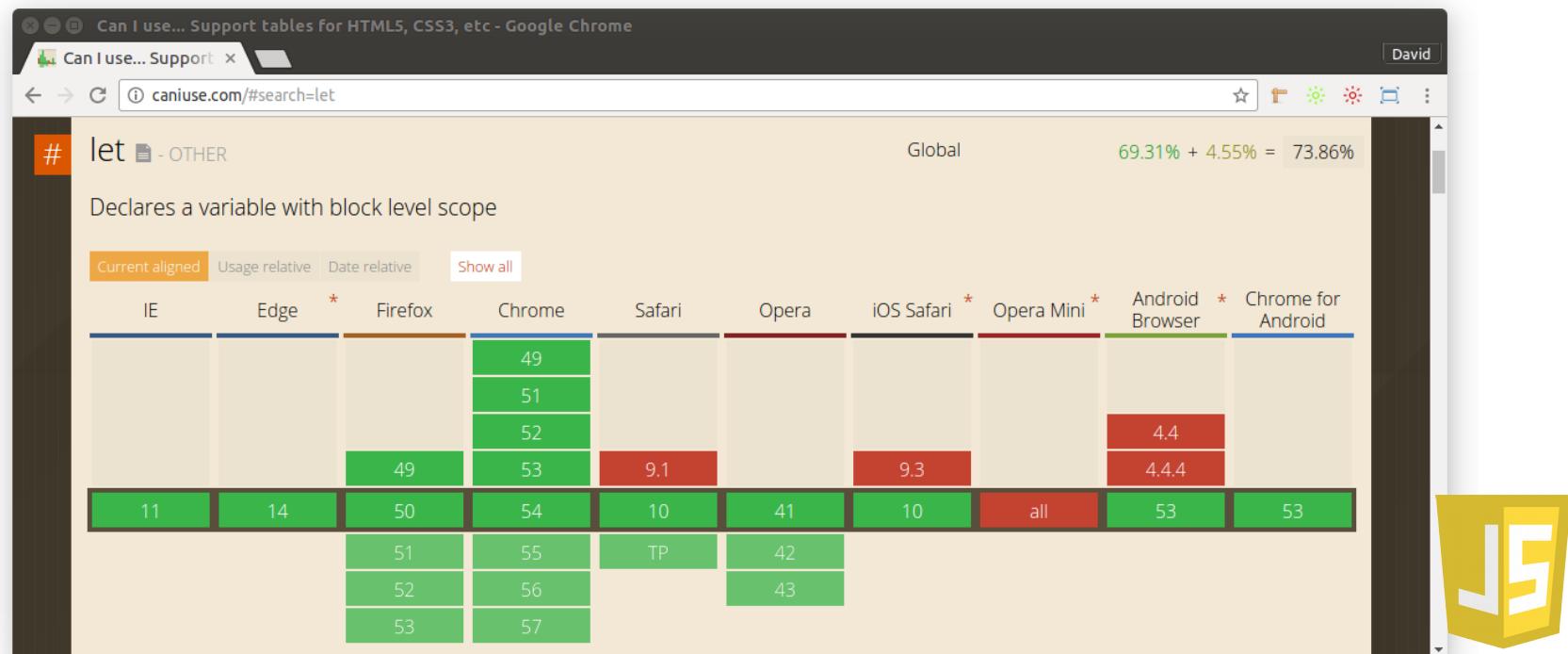
Node.Js

- Além de ser a principal linguagem para desenvolvimento front-end, o JavaScript também tem ganhado espaço no mundo back-end através do **Node.Js**



ECMAScript 6

- Nas próximas aulas faremos uma breve apresentação do ES6, versão do ECMAScript lançada em 2015
- No entanto, por ser uma versão mais recente do ES, muitos browsers ainda não a suportam completamente



ECMAScript 6

- Neste curso faremos uma breve apresentação do ES6 versão 6.
- No final da aula haverá muitas perguntas.

Babel · The compiler for writing next generation JavaScript - Google Chrome

babeljs.io

BABEL

Babel is a JavaScript compiler.

Use next generation JavaScript, today.

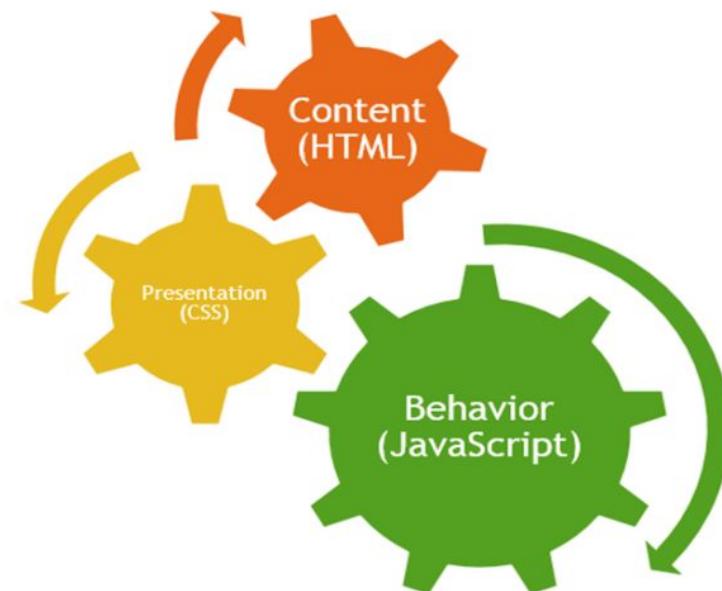
Podemos usar a biblioteca Babel para converter código ES6 para códigos de versões anteriores do JavaScript.

JavaScript Engine	Support Level
4.4	Red
4.4.4	Red
53	Green
53	Green

JS

Inserindo JS em sua página

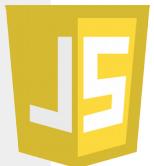
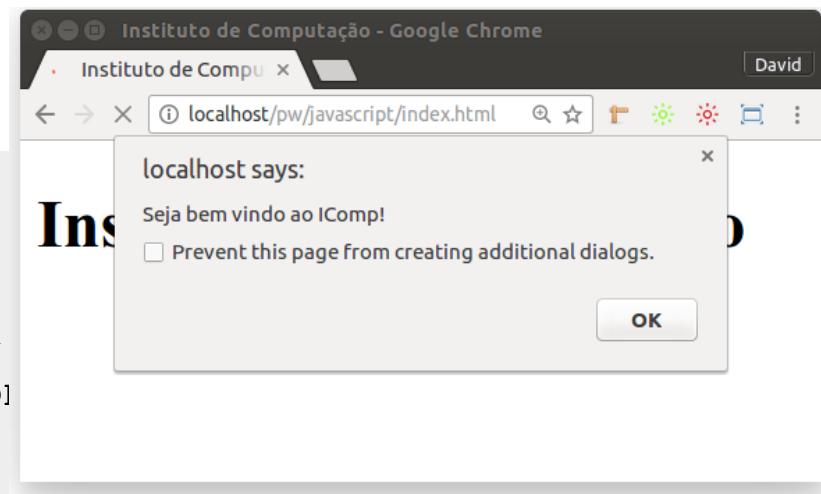
- Para adicionar códigos JavaScript à uma página devemos usar a tag `<script>`
- Essa inserção pode ser feita através de dois métodos:
 - Método embardado
 - Método externo



Método Embarcado

- Cria-se uma tag <script> e então, coloca-se o código JavaScript dentro dessa tag

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Instituto de Co
  </head>
  <body>
    <h1>Instituto de Computação</h1>
    <script type="text/javascript">
      alert('Seja bem vindo IComp!');
    </script>
  </body>
</html>
```

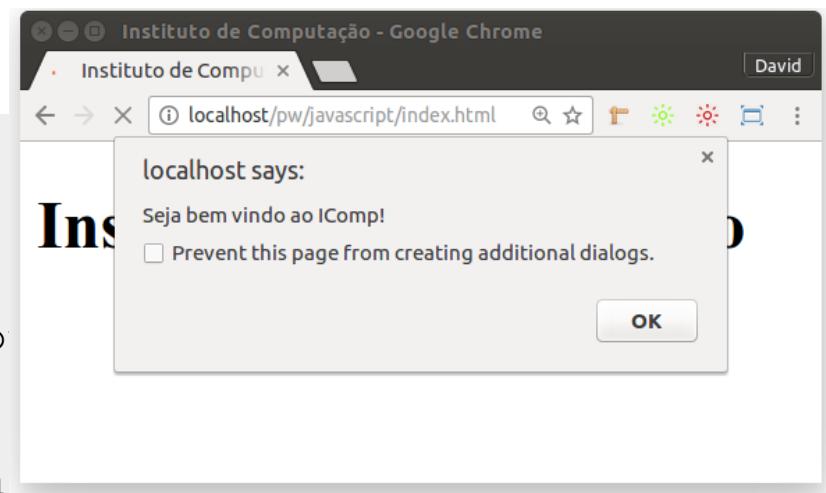


Método Externo

- Cria-se uma tag <script> e então usa-se o atributo **src** para definir o path do arquivo JavaScript

- **arquivo index.html:**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Instituto de Comp
  </head>
  <body>
    <h1>Instituto de Computa
    <script type="text/javascript" src="script.js">
    </script>
  </body>
</html>
```



- **arquivo script.js:**

```
alert('Seja bem vindo IComp!');
```



Onde colocamos o <script>?

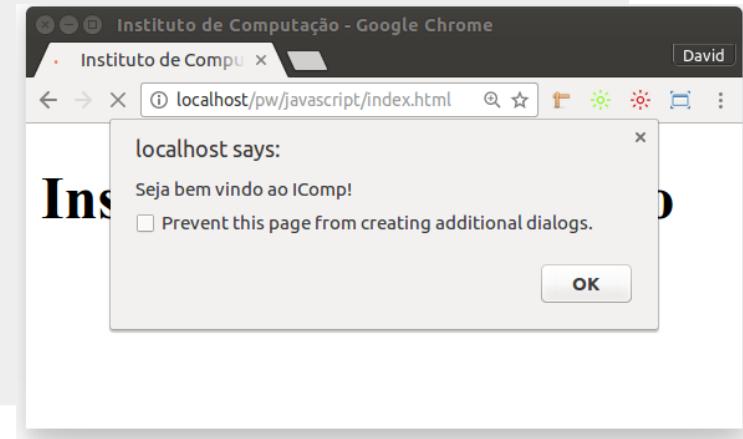
- No cabeçalho do documento HTML
 - Neste caso ele é lido antes do documento HTML ser renderizado, e o código será executado imediatamente
- No corpo do documento HTML
 - Neste caso ele é lido enquanto o documento HTML é renderizado. Quando encontra o <script>, o browser para a renderização para interpretar o código JavaScript
- Imediatamente antes de </body>
 - Neste caso, o código é executado após todo o conteúdo HTML ter sido renderizado pelo browser



Onde colocamos o <script>?

- Pode-se colocar várias tags <script> em diferentes regiões de uma mesma página

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Instituto de Computação</title>
        <script type="text/javascript">
            function bemvindo() {
                alert('Seja bem vindo ao IComp!');
            }
        </script>
    </head>
    <body>
        <h1>Instituto de Computação</h1>
        <script type="text/javascript">
            bemvindo();
        </script>
    </body>
</html>
```

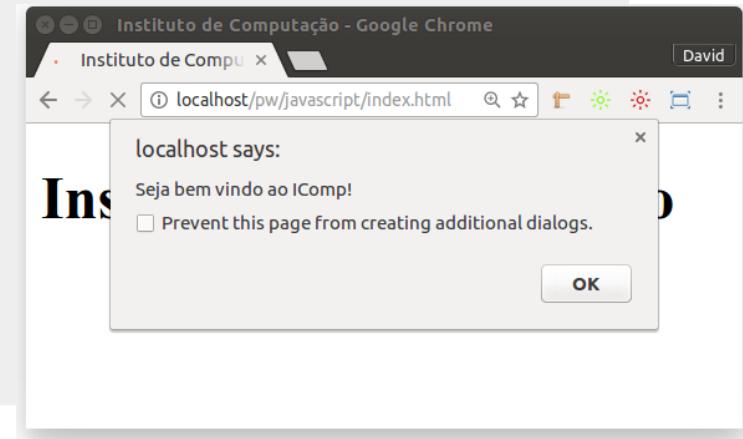


Onde colocamos o <script>?

- Pode-se colocar várias tags <script> em diferentes regiões de uma mesma página

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Instituto de Computação</title>
    <script type="text/javascript">
      function bemvindo(local) {
        alert('Seja bem vindo ao ' + local + '!');
      }
    </script>
  </head>
  <body>
    <h1>Instituto de Computação</h1>
    <script type="text/javascript">
      bemvindo("IComp");
    </script>
  </body>
</html>
```

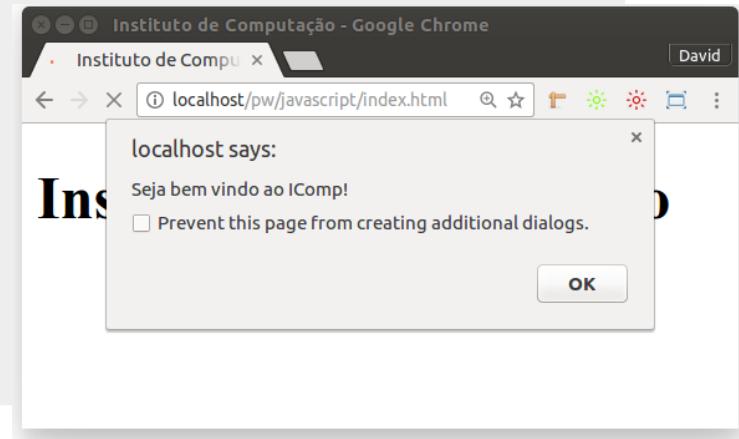
Sinal de
concatenação



Onde colocamos o <script>?

- Pode-se colocar várias tags <script> em diferentes regiões de uma mesma página

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Instituto de Computação</title>
        <script type="text/javascript">
            function bemvindo(local) {
                return 'Seja bem vindo ao ' + local + '!';
            }
        </script>
    </head>
    <body>
        <h1>Instituto de Computação</h1>
        <script type="text/javascript">
            alert(bemvindo("IComp"));
        </script>
    </body>
</html>
```



Onde colocamos o <script>?

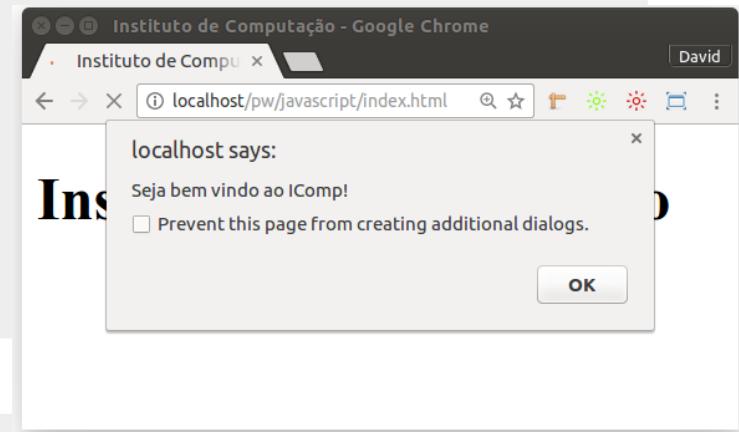
- Pode-se colocar várias tags <script> em diferentes regiões de uma mesma página

- arquivo index.html:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Instituto de Computação</title>
    <script type="text/javascript" src="bemvindo.js"></script>
  </head>
  <body>
    <h1>Instituto de Computação</h1>
    <script type="text/javascript">
      bemvindo("IComp");
    </script>
  </body>
</html>
```

- arquivo bemvindo.js:

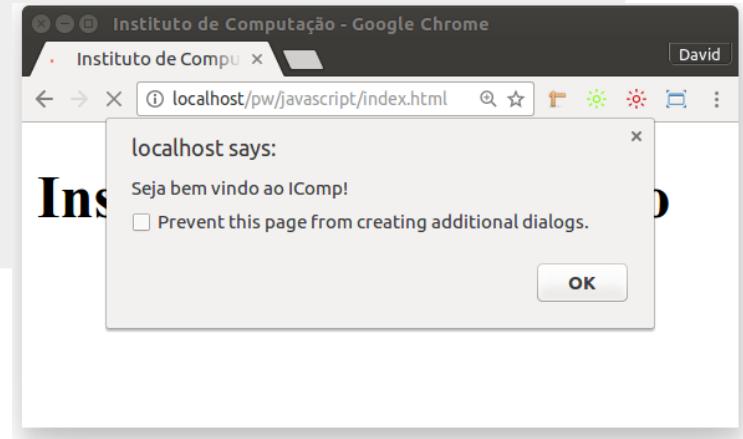
```
function bemvindo(local) {
  alert('Seja bem vindo ao ' + local + '!');
}
```



Onde colocamos o <script>?

- Pode-se colocar várias tags <script> em diferentes regiões de uma mesma página

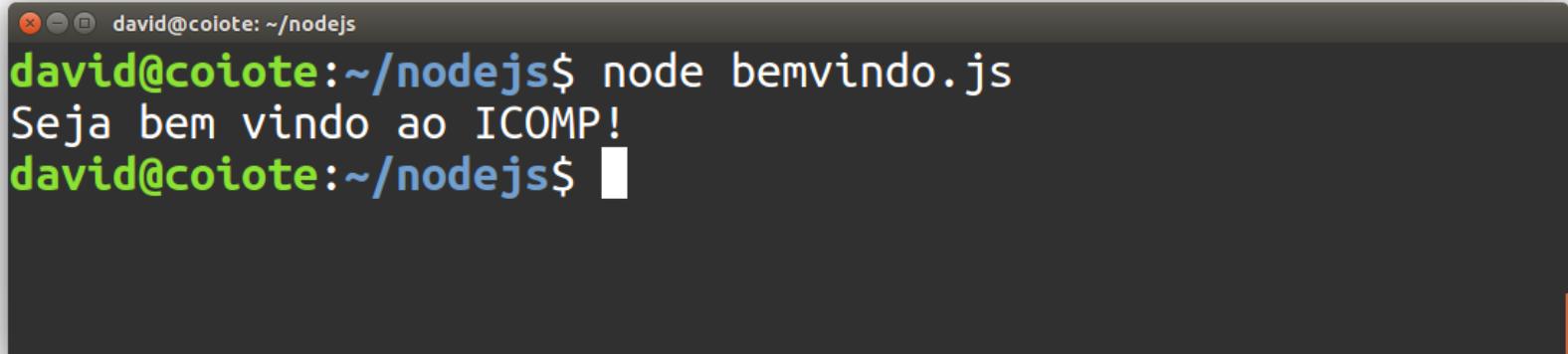
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Instituto de Computação</title>
    <script type="text/javascript">
      function bemvindo(local) {
        alert('Seja bem vindo ao ' + local + '!');
      }
    </script>
  </head>
  <body onload="bemvindo('IComp')">
    <h1>Instituto de Computação</h1>
  </body>
</html>
```



Execução via Node.Js

- Os scripts JS também podem ser executados através do Node.Js instalado em seu Sistema Operacional
 - **arquivo bemvindo.js:**

```
function bemvindo(local) {  
    console.log('Seja bem vindo ao ' + local + '!');  
}  
  
bemvindo("ICOMP");
```



A screenshot of a terminal window titled "david@coiote: ~/nodejs". The window shows the command "node bemvindo.js" being run, followed by the output "Seja bem vindo ao ICOMP!". The terminal has a dark background with light-colored text.

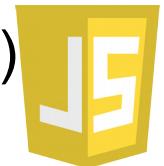
```
david@coiote:~/nodejs$ node bemvindo.js  
Seja bem vindo ao ICOMP!  
david@coiote:~/nodejs$ █
```

Valores e Tipos

- **Tipagem dinâmica:** Tipos são associados a valores, e não a variáveis

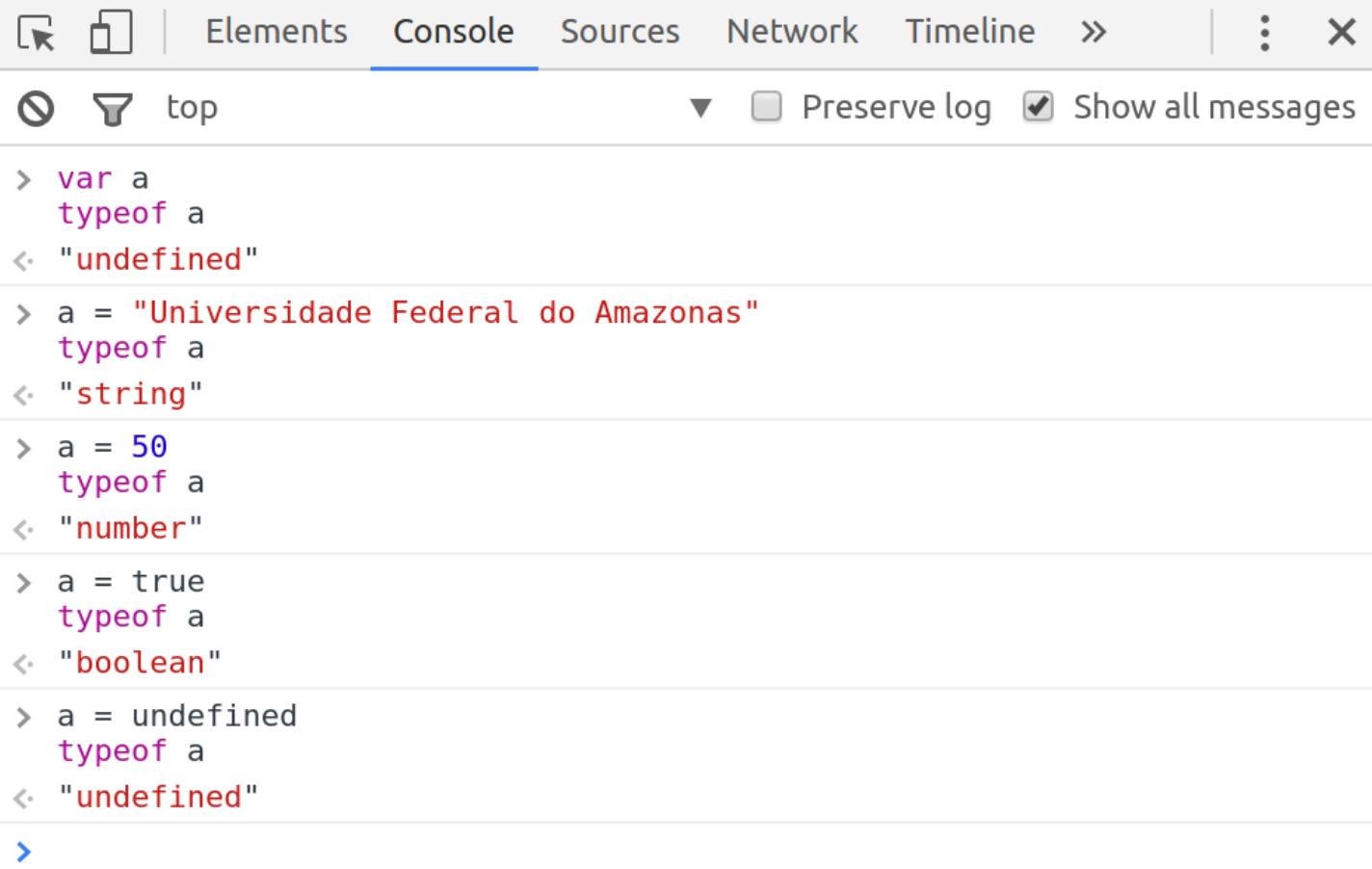
```
var i = 1;  
i = "a string";  
i = new Date();
```

- Os tipos de valores presentes na linguagem são:
 - number (inteiros ou pontos flutuantes)
 - string (pode-se usar aspas simples ou duplas)
 - boolean
 - object
 - null (para objetos) e undefined (para valores primitivos)



Valores e Tipos

- O tipo de uma variável em dado instante pode ser checado através da função **typeof**



The screenshot shows the 'Console' tab of a browser's developer tools. It displays several examples of the `typeof` operator being used on different variables:

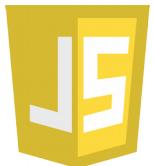
- `> var a
typeof a`
Output: `< "undefined"`
- `> a = "Universidade Federal do Amazonas"
typeof a`
Output: `< "string"`
- `> a = 50
typeof a`
Output: `< "number"`
- `> a = true
typeof a`
Output: `< "boolean"`
- `> a = undefined
typeof a`
Output: `< "undefined"`
- `>`



Tipo Number

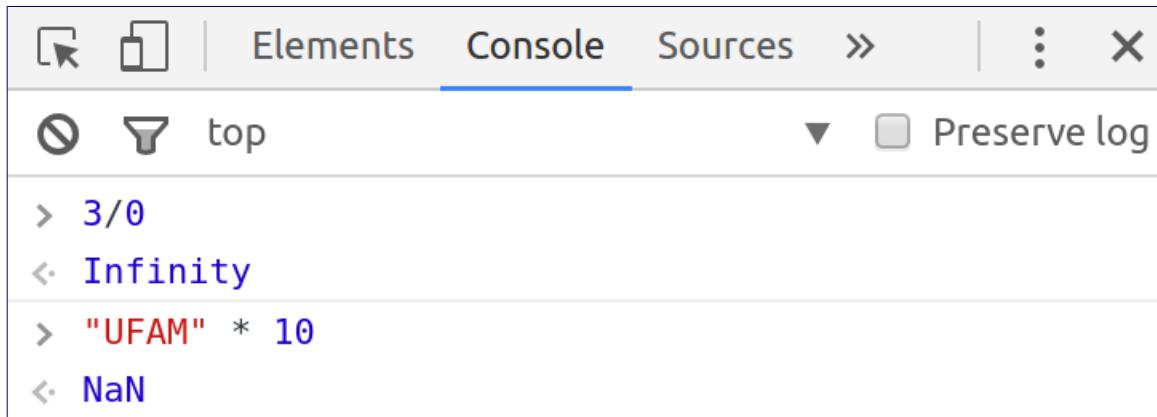
- Number é um tipo de valor que representa números inteiros e números de ponto flutuante
 - O tipo numérico (inteiro ou real) é definido pelo valor assumido pela variável
- Não existe tipo inteiro separado

```
<script type="text/javascript">
    var diasDaSemana = 7;
    var pi = 3.1415;
    var hexValue = 0xFFFF;
</script>
```



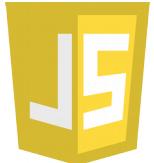
Valores Numéricos Especiais

- **Infinity**
- **NaN**
 - Not a number
- **Number.MAX_VALUE**
 - Maior número positivo
- **Number.MIN_VALUE**
 - Menor número positivo maior que zero



The screenshot shows the Chrome DevTools Console tab. The console output is as follows:

```
> 3/0
<- Infinity
> "UFAM" * 10
<- NaN
```



Padrão IEEE 754

- Valores reais de precisão simples e dupla



- Cuidado com os resíduos binários!

```
david@coyote: ~
> 0.1 + 0.2
0.30000000000000004
>
```



Objeto Math

- **Math** é um objeto pré-definido que tem propriedades e métodos para constantes e funções matemáticas
 - Na imagem ao lado, usamos o console do **NodeJS** para listar as propriedades e métodos de Math

```
david@david-OptiPlex-5090: ~$ nodejs
> Math.
Math.__defineGetter__          Math.__defineSetter__
Math.__lookupGetter__          Math.__lookupSetter__
Math.constructor               Math.hasOwnProperty
Math.isPrototypeOf             Math.propertyIsEnumerable
Math.toLocaleString            Math.toString
Math.valueOf

Math.E                         Math.LN10
Math.LN2                        Math.LOG10E
Math.LOG2E                      Math.PI
Math.SQRT1_2                    Math.SQRT2
Math.abs                        Math.acos
Math.asin                       Math.atan
Math.atan2                      Math.ceil
Math.cos                        Math.exp
Math.floor                      Math.log
Math.max                        Math.min
Math.pow                         Math.random
Math.round                       Math.sin
Math.sqrt                       Math.tan

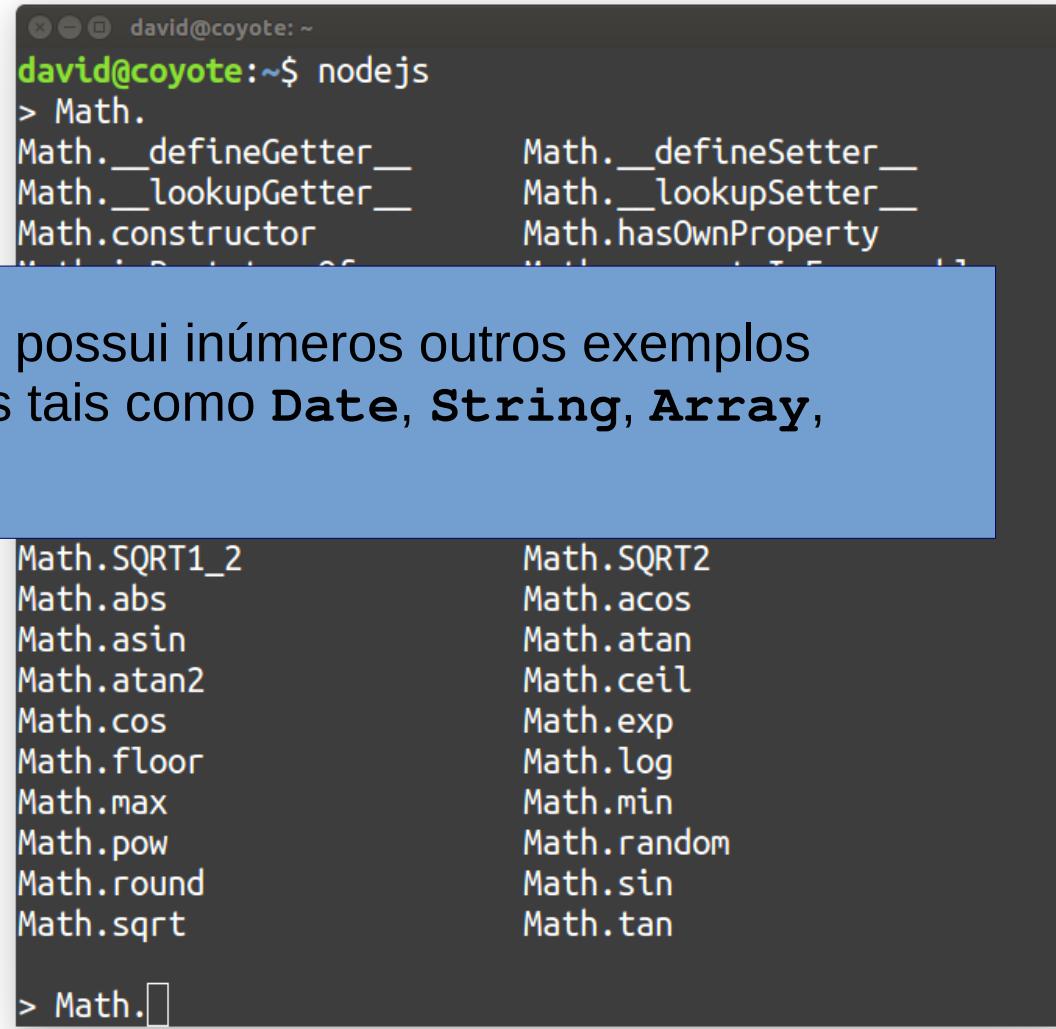
> Math.[]
```

Objeto Math

- **Math** é um objeto pré-definido que tem propriedades e métodos para...

A linguagem JavaScript possui inúmeros outros exemplos de objetos pré-definidos tais como **Date**, **String**, **Array**, **Number**, **JSON**, etc.

- Na imagem ao lado, usamos o console do **NodeJS** para listar as propriedades e métodos de Math



```
david@coyote:~$ nodejs
> Math.
Math._defineGetter_
Math._lookupGetter_
Math.constructor
Math.E
Math.LN2
Math.LN10
Math.LOG2E
Math.LOG10E
Math.PI
Math.SQRT1_2
Math.abs
Math.asin
Math.atan2
Math.cos
Math.floor
Math.max
Math.pow
Math.round
Math.sqrt
Math.SQRT2
Math.acos
Math.atan
Math.ceil
Math.exp
Math.log
Math.min
Math.random
Math.sin
Math.tan
> Math.[]
```

Tipo String

- Valores string são usados para manipular um pedaço de texto armazenado
- As seguintes declarações são iguais:

```
<script type="text/javascript">
    var s = 'UFAM';
    var s = "UFAM";
    var s = new String("UFAM");
</script>
```

Não há razão para
fazer desse jeito



Tipo String

- Após a declaração de uma string, a variável passa a ter um conjunto de métodos e propriedades (imutáveis)

```
<script type="text/javascript">
    var s = 'UFAM';
</script>
```

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



Tipo String

- Após a declaração de uma **string**, a variável passa a ter um conjunto de métodos e propriedades

```
david@coyote:~$ nodejs
> s = 'UFAM'
'UFAM'
> s.
  s.__defineGetter__      s.__defineSetter__
  s.__lookupGetter__     s.__lookupSetter__
  s.constructor          s.hasOwnProperty
  s.isPrototypeOf        s.propertyIsEnumerable
  s.toLocaleString       s.toString
  s.valueOf

  s.anchor                s.big
  s.blink                 s.bold
  s.charAt                s.charCodeAt
  s.concat                s.fixed
  s.fontcolor              s.fontsize
  s.indexOf                s.italics
  s.lastIndexOf           s.length
  s.link                  s.localeCompare
  s.match                 s.replace
  s.search                s.slice
  s.small                 s.split
  s.strike                s.sub
  s.substr                s.substring
  s.sup                   s.toLocaleLowerCase
  s.toLocaleUpperCase     s.toLowerCase
  s.toUpperCase           s.trim
  s.trimLeft              s.trimRight
```

Valores Booleanos

- Uma variável do tipo primitivo booleano pode assumir os valores `true` e `false` (case sensitive)
- Operadores que podem produzir valores booleanos:
 - Operadores lógicos binários: `&&` (And), `||` (Or)
 - Operadores prefixos lógicos: `!` (Not)
 - Operadores de comparação:
 - Operadores de igualdade: `====` , `!==` , `==` , `!=`
 - Operadores de ordem: `>` , `>=` , `<` , `<=`

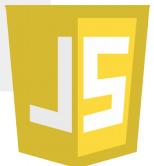


Valores Booleanos

- A linguagem suporta quatro tipos de operadores lógicos de igualdade: `==`, `====`, `!=`, e `!==`
- A diferença entre `==` e `====` é que `==` compara apenas os valores das variáveis, enquanto `====` compara os valores e os tipos de valores
 - O operador `====` é chamado de igualdade estrita
 - O operador `!==` é chamado de desigualdade estrita

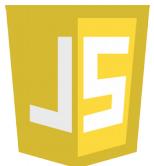
```
var a = "42";
var b = 42;

a == b;           // true
a === b;          // false
```



Valores Booleanos

- Os valores abaixo são considerados falsos pela linguagem JavaScript:
 - " " (string vazia)
 - 0, -0, **NaN** (not a number)
 - **null**, **undefined**
 - **false**
- Qualquer outro valor é considerado verdadeiro pela linguagem



Avaliação de curto-circuito

- Expressões lógicas como as do código abaixo são avaliadas da esquerda para a direita

```
A && B  
A || B
```

- Desta forma, o segundo argumento só é executado ou avaliado se o primeiro argumento não for suficiente para determinar o valor da expressão
- Note que:

```
false && qualquercoisa = false.  
true || qualquercoisa = true.
```



Avaliação de curto-circuito

- Expressões lógicas como as do código abaixo são avaliadas da esquerda para a direita

```
A && B  
A || B
```

- Uma vantagem do uso de avaliações em curto circuito é a condicionar a execução de uma função ao retorno de outra função:
- primeiraFuncao () || segundaFuncao ()

```
false && qualquercoisa = false.  
true || qualquercoisa = true.
```

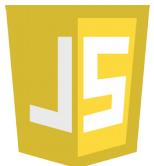


Exercício

- Qual é o valor de **v** após a execução do código abaixo?

```
var v = v || 0;
```

- Experimente os seguintes valores iniciais para **v**: **100**, **false**, ou **null**.



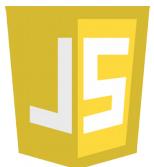
Condicionais

- O comando **if** possui uma cláusula opcional **else** que é executada dependendo de uma condição booleana:

```
if (myvar === 0) // then  
  
if (myvar === 0) {  
    // then  
}  
  
if (myvar === 0) {  
    // then  
} else {  
    // else  
}
```

```
if (myvar === 0) {  
    // then  
} else if (myvar === 1) {  
    // else-if  
} else if (myvar === 2) {  
    // else-if  
} else {  
    // else  
}
```

- A linguagem JavaScript não possui atalhos **elseif** ou **elif**



Condicionais

- O comando **if** possui uma cláusula opcional **else** que é executada dependendo de uma condição booleana:

```
if (myvar === 0) // then    if (myvar === 0) {
```

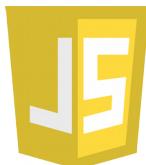
JavaScript também suporta **operadores condicionais ternários**, através da seguinte sintaxe:

```
test ? expression1 : expression2
```

Exemplo de aplicação:

```
var now = new Date();
var saudacao = "Boa " + ((now.getHours()>=12) ? "tarde" :"dia");
                    // else
}
```

- A linguagem JavaScript não possui atalhos **elseif** ou **elif**



Condicionais

- A linguagem suporta declarações **switch**, onde o valor de uma variável define que **case** é executado

```
switch (fruit) {  
    case 'banana':  
        // ...  
        break;  
    case 'apple':  
        // ...  
        break;  
    default:  
        // ...  
}
```

- O operando definido após cada case é comparado com o parâmetro de switch via **==**



Loops

- A linguagem JavaScript suporta os mesmos tipos de loops de linguagens como C e Java

- **for**

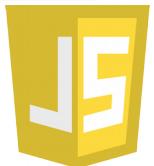
```
for (var i=0; i < arr.length; i++) {  
    console.log(arr[i]);  
}
```

- **while**

```
var i = 0;  
while (i < arr.length) {  
    console.log(arr[i]);  
    i++;  
}
```

- **do...while**

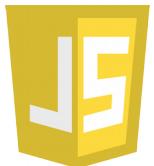
```
var i = 0;  
do {  
    console.log(arr[i]);  
    i++;  
} while (i <= arr.length);
```



Função `document.write()`

- Podemos usar as funções `document.write()` e `document.writeln()` para imprimir na página Web

```
document.write("<h1>Instituto de Computação</h1>");
```

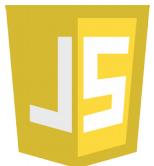


Comentários

- A linguagem JavaScript suporta os mesmos tipos de comentários de linguagens como C e Java

```
// .....
```

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



Exercício

- Faça um página contendo a tabela de multiplicação dos números 1 a 10. O conteúdo dessa página deve ser inteiramente gerado através de JavaScript (o elemento body da página deve estar vazio). Use código CSS para manter o estilo exatamente igual ao da página ao lado.

The screenshot shows a multiplication table generator. The page has a header 'Instituto de Computação - Google Chrome' and a URL 'localhost/pw/index2.html'. On the right, a red oval contains the text 'github JS1'. Below the browser window is a yellow JS logo.

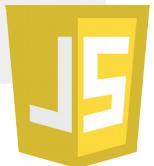
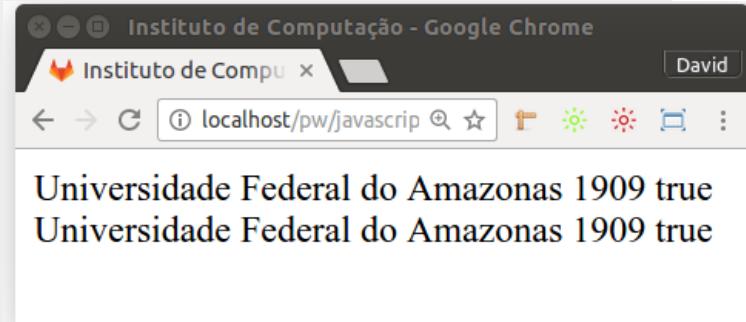
Produtos de 1		Produtos de 2		Produtos de 3		Produtos de 4		Produtos de 5	
1x1	1	2x1	2	3x1	3	4x1	4	5x1	5
1x2	2	2x2	4	3x2	6	4x2	8	5x2	10
1x3	3	2x3	6	3x3	9	4x3	12	5x3	15
1x4	4	2x4	8	3x4	12	4x4	16	5x4	20
1x5	5	2x5	10	3x5	15	4x5	20	5x5	25
1x6	6	2x6	12	3x6	18	4x6	24	5x6	30
1x7	7	2x7	14	3x7	21	4x7	28	5x7	35
1x8	8	2x8	16	3x8	24	4x8	32	5x8	40
1x9	9	2x9	18	3x9	27	4x9	36	5x9	45
1x10	10	2x10	20	3x10	30	4x10	40	5x10	50

Produtos de 6		Produtos de 7		Produtos de 8		Produtos de 9		Produtos de 10	
6x1	6	7x1	7	8x1	8	9x1	9	10x1	10
6x2	12	7x2	14	8x2	16	9x2	18	10x2	20
6x3	18	7x3	21	8x3	24	9x3	27	10x3	30
6x4	24	7x4	28	8x4	32	9x4	36	10x4	40
6x5	30	7x5	35	8x5	40	9x5	45	10x5	50
6x6	36	7x6	42	8x6	48	9x6	54	10x6	60
6x7	42	7x7	49	8x7	56	9x7	63	10x7	70
6x8	48	7x8	56	8x8	64	9x8	72	10x8	80
6x9	54	7x9	63	8x9	72	9x9	81	10x9	90
6x10	60	7x10	70	8x10	80	9x10	90	10x10	100

Tipo objeto

- Um objeto é uma **coleção dinâmica de chaves e valores** de quaisquer tipos de dados

```
var ufam = {  
    nome: "Universidade Federal do Amazonas",  
    fundacao: 1909,  
    ativa: true  
};  
  
document.writeln(ufam.nome);  
document.writeln(ufam.fundacao);  
document.writeln(ufam.ativa);  
  
document.writeln("<br>");  
  
document.writeln(ufam["nome"]);  
document.writeln(ufam["fundacao"]);  
document.writeln(ufam["ativa"]);
```



Tipo objeto

- Um objeto é uma **coleção dinâmica de chaves e valores** de quaisquer tipos de dados

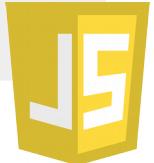
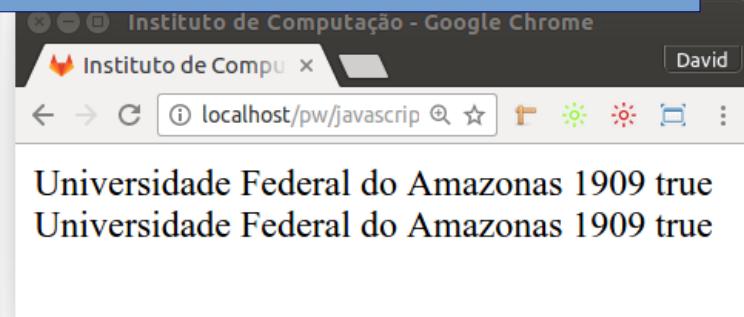
As propriedades podem ser acessadas através da **notação ponto** (`ufam.nome`) ou através da **notação colchetes** (`ufam["nome"]`).

A notação ponto é mais simples e fácil de ler, e portanto é preferível usá-la quando possível

```
document.writeln(ufam.nome);
document.writeln(ufam.fundacao);
document.writeln(ufam.ativa);

document.writeln("<br>");

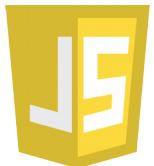
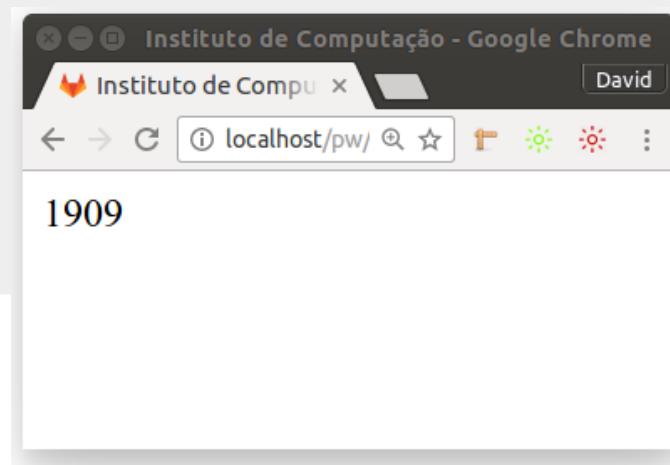
document.writeln(ufam["nome"]);
document.writeln(ufam["fundacao"]);
document.writeln(ufam["ativa"]);
```



Tipo objeto

- A notação colchetes pode ser conveniente para acessar o valor de uma propriedade usando uma variável:

```
var ufam = {  
    nome: "Universidade Federal do Amazonas",  
    fundacao: 1909,  
    ativa: true  
};  
  
var ano = "fundacao";  
  
document.write(ufam[ano]);
```



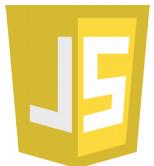
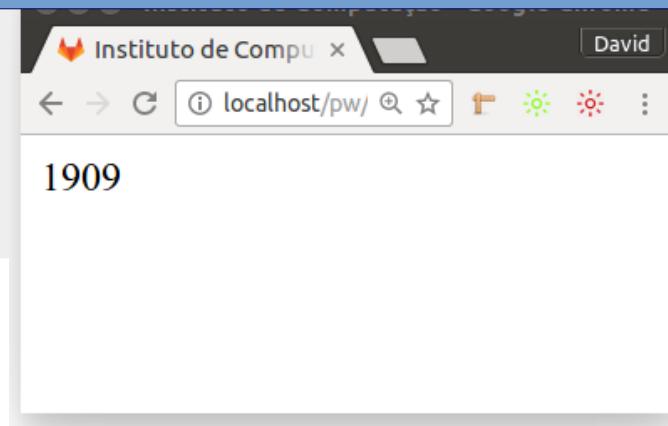
Tipo objeto

- A notação colchetes pode ser conveniente para acessar o valor de uma propriedade usando uma variável:

A notação de colchetes também é obrigatória quando o nome da propriedade for numérico ou quando contiver espaços:

```
objeto["10"] = 100;  
objeto["algum nome"] = "algum valor";
```

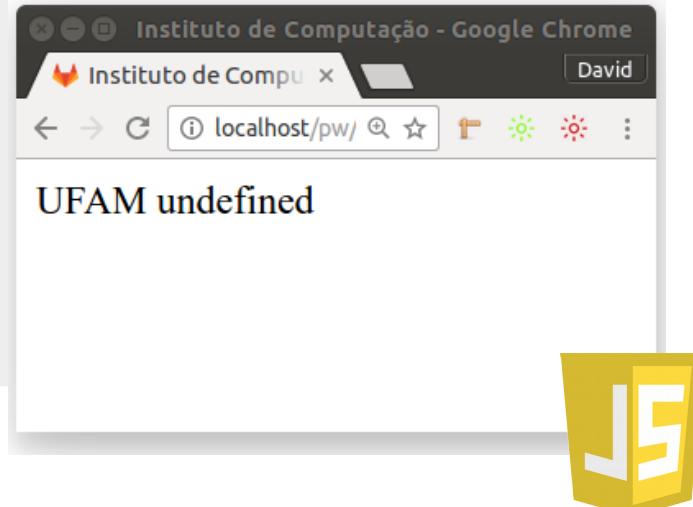
```
,  
  
var ano = "fundacao";  
  
document.write(ufam[ano]);
```



Tipo objeto

- É possível adicionar ou remover propriedades a qualquer momento
- Para apagar uma propriedade ou método de um objeto, podemos usar o operador **delete**

```
var ufam = {  
    nome: "Universidade Federal do Amazonas",  
    fundacao: 1909,  
    ativa: true  
};  
  
ufam.sigla = "UFAM";  
delete ufam.fundacao;  
  
document.writeln(ufam.sigla);  
document.writeln(ufam.fundacao);
```

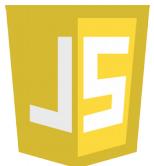


Tipo objeto

- Para definir um método dentro do objeto, usamos a palavra-chave **function**
- Usamos a palavra chave **this** para referenciar métodos e propriedades internos do objeto

```
var ufam = {
    nome: "Universidade Federal do Amazonas",
    fundacao: 1909,
    ativa: true,
    imprime: function () {
        document.writeln(this.nome);
    },
};

ufam.imprime();
```

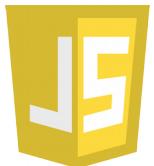
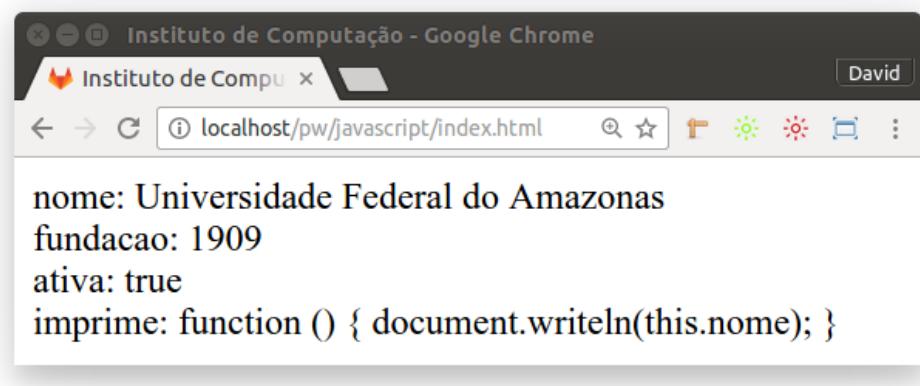


Tipo objeto

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

```
for (var prop in ufam) {  
    document.writeln(prop +": "+ ufam[prop] + "<br>");  
}
```

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

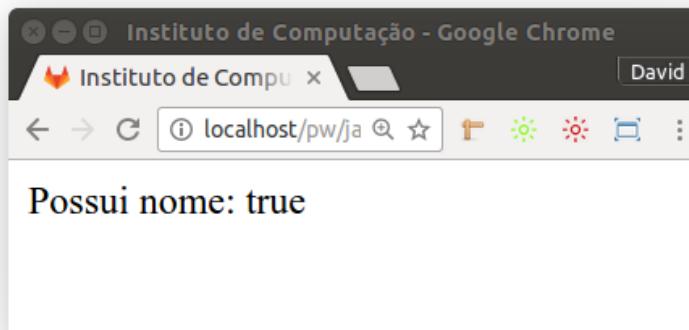


Tipo objeto

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

```
var ufam = {
    nome: "Universidade Federal do Amazonas",
    fundacao: 1909,
    ativa: true,
    imprime: function () {
        console.log(this.name);
    },
};

var possuiNome = "nome" in ufam;
document.writeln("Possui nome: " + possuiNome);
```

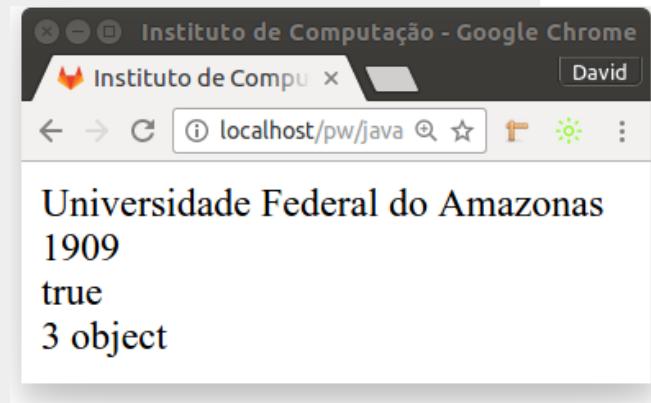


Arrays

- Em JavaScript, um array é um objeto cujos valores não são referenciados por propriedades/chaves, mas por **posições numéricas indexadas**

```
var arr = [
    "Universidade Federal do Amazonas",
    1909,
    true
];

document.writeln(arr[0] + "<br>");
document.writeln(arr[1] + "<br>");
document.writeln(arr[2] + "<br>");
document.writeln(arr.length);
document.writeln(typeof arr);
```

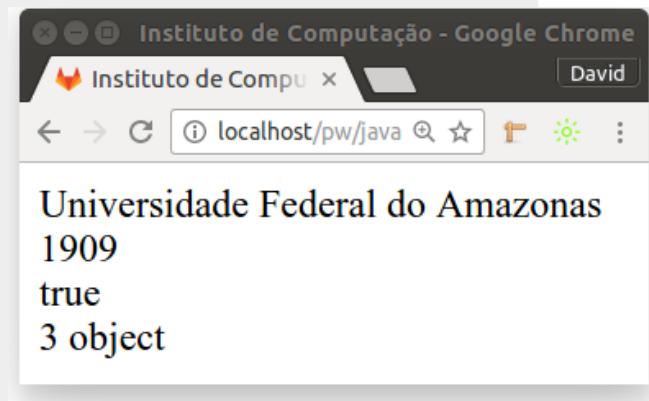


Arrays

- Em JavaScript, um array é um objeto cujos valores não
são referenciados por nomes individuais (chaves) mas por

Como os arrays são objetos especiais (conforme visto pelo `typeof`),
eles também podem ter propriedades, incluindo a propriedade `length`
(que é atualizada automaticamente)

```
1909,  
true  
];  
  
document.writeln(arr[0] + "<br>");  
document.writeln(arr[1] + "<br>");  
document.writeln(arr[2] + "<br>");  
document.writeln(arr.length);  
document.writeln(typeof arr);
```



Arrays

- Os códigos abaixo fazem exatamente a mesma coisa

```
var arr = [
    "Universidade Federal do Amazonas",
    1909,
    true
];
```

```
var arr = [];
arr[0] = "Universidade Federal do Amazonas",
arr[1] = 1909,
arr[2] = true
```

```
var arr = new Array();
arr[0] = "Universidade Federal do Amazonas",
arr[1] = 1909,
arr[2] = true
```



Arrays

- Os códigos abaixo fazem exatamente a mesma coisa

```
var arr = [  
    "Universidade Federal do Amazonas",  
    1909,
```

O vetor cresce dinamicamente a medida que novos elementos são inseridos

Os ítems dos vetores podem ser de tipos diferentes

```
arr[0] = "Universidade Federal do Amazonas",  
arr[1] = 1909,  
arr[2] = true
```

```
var arr = new Array();  
arr[0] = "Universidade Federal do Amazonas",  
arr[1] = 1909,  
arr[2] = true
```

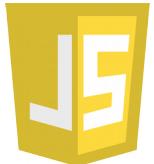


Arrays

- Todos os arrays possuem um método **forEach**, que pode ser usado para percorrer os elementos do array

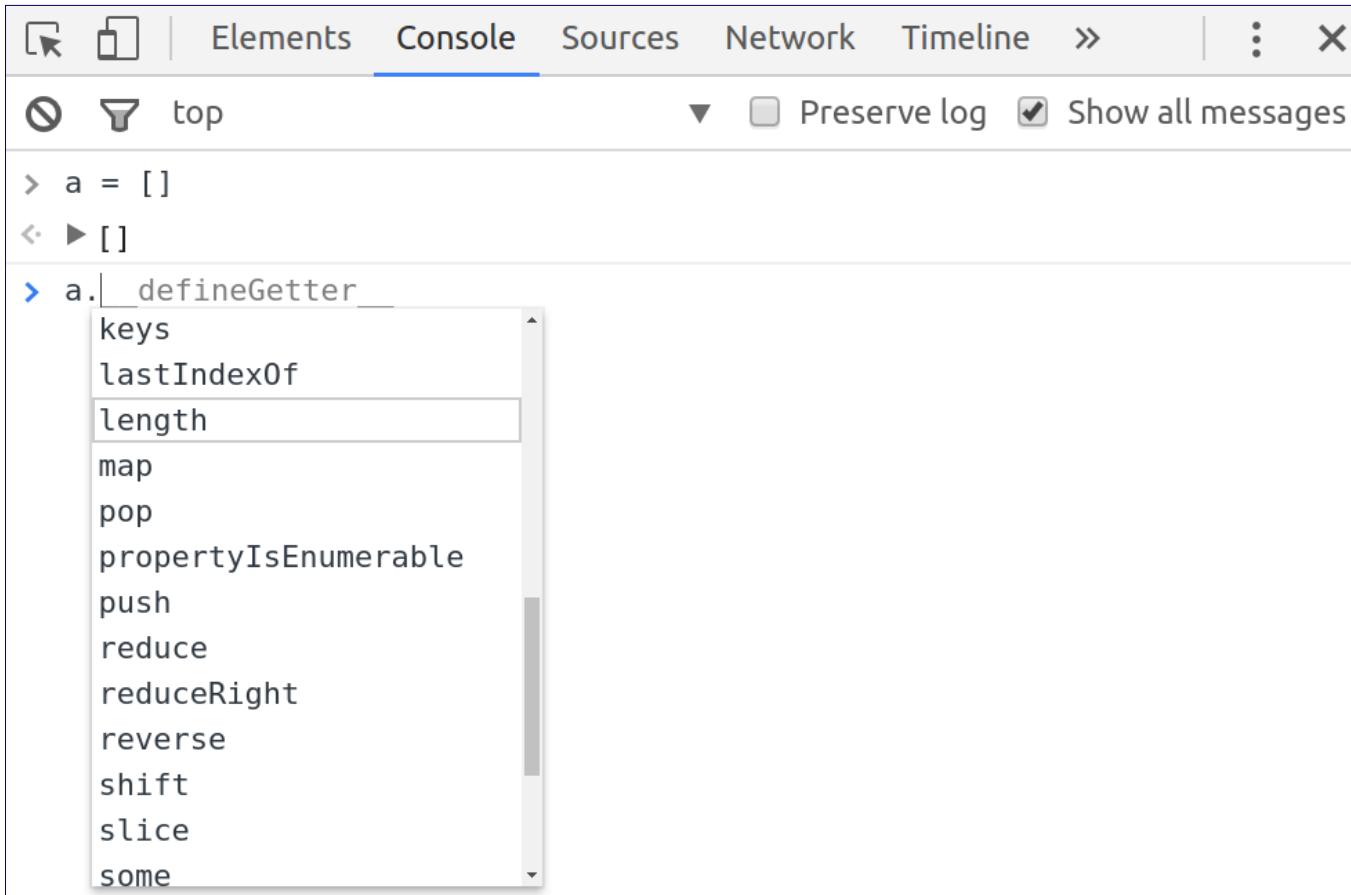
```
var arr = [
    "Universidade Federal do Amazonas",
    1909,
    true
];

arr.forEach(function (valor) {
    document.writeln(valor + "<br>");
});
```



Arrays

- Os arrays possuem outras propriedades e métodos, que podem ser vistas pelo terminal JavaScript ou Node.js



The screenshot shows the Chrome DevTools console interface. The 'Console' tab is active. In the input field, the code 'a = []' has been entered, resulting in the output 'top'. Below the input field, the cursor is positioned after the dot in 'a.', and a dropdown menu lists various array methods: __defineGetter__, keys, lastIndexof, length, map, pop, propertyIsEnumerable, push, reduce, reduceRight, reverse, shift, slice, and some. The 'length' method is currently selected in the dropdown.



Arrays

- Exemplos de métodos bastante úteis presentes nos objetos do tipo Array

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



Arrays

- Exemplos de métodos bastante úteis presentes nos objetos do tipo Array

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

Funções JavaScript

- Antes da versão ES7 (de 2015), a linguagem JavaScript não possuia classes, métodos, construtores e módulos
- Em JavaScript, a existência de todas essas construções podem ser simuladas através de Funções
- Em JavaScript, as funções também são objetos:

```
function foo() {  
    return 50;  
}  
  
foo.bar = "UFAM/IComp";  
  
document.writeln(typeof foo);  
document.writeln(typeof foo());  
document.writeln(typeof foo.bar);
```



Funções de Primeira Classe

- Uma linguagem só pode ser considerada funcional se tratar suas funções como elementos de primeira classe
 - Isto significa que as funções em JavaScript são tipos especiais de objetos que podem fazer tudo o que um objeto normal pode fazer
- Ações permitidas por funções de primeira classe:
 - Atribuir uma função a uma variável
 - Retornar uma função como valor de outra função
 - Passar uma função como parâmetro para outra função
 - Armazenar uma função em uma estrutura de dados



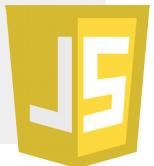
Funções de Primeira Classe

- Atribuir uma função a uma variável

```
var imprimir = function(str) {  
    console.log(str);  
}  
  
imprimir();
```

- Retornar uma função como valor de outra função

```
function funcaoA(x) {  
    return function funcaoB(x) { return x * 2 };  
}  
  
var resposta = funcaoA();  
resposta(50); // 100
```



Funções de Primeira Classe

- Passar uma função como parâmetro para outra função

```
var produto = {  
    nome: 'sapato',  
    preco: 150.0  
}  
  
var formulaImpostoA = function (preco) {  
    return preco * 0.1;  
}  
  
var formulaImpostoB = function (preco) {  
    return preco * 0.2;  
}  
  
var calculaPreco = function (produto, formulaImpostoA) {  
    return produto.preco + formulaImpostoA(produto.preco)  
}  
  
calculaPreco(produto, formulaImpostoA); // 165  
calculaPreco(produto, formulaImpostoB); // 180
```

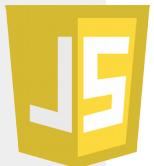
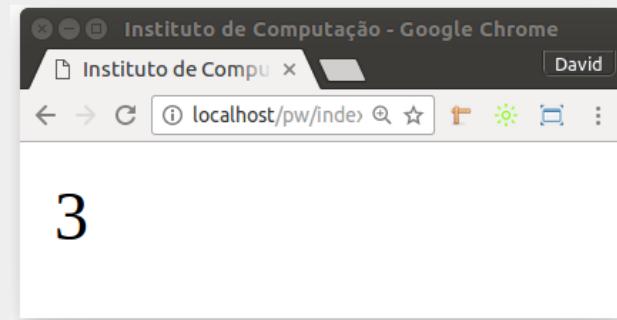


Funções de Primeira Classe

- Quando passamos uma função A como argumento para uma função B, e B executa A, então chamamos a função A de **callback**

```
function soma(a, b) {  
    return a() + b();  
}  
  
function um() {  
    return 1;  
}  
  
function dois() {  
    return 2;  
}  
  
valor = soma(um, dois);  
document.writeln(valor);
```

Callbacks

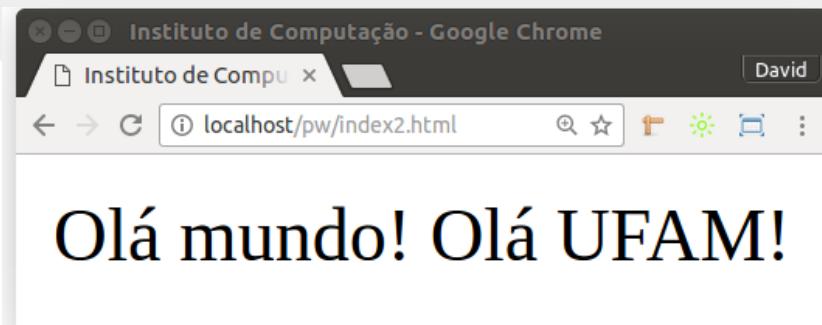


Funções de Primeira Classe

- **Funções imediatas:** também é possível criar funções que são executadas imediatamente após a sua definição
- Tais funções são úteis quando queremos fazer algum processamento imediato sem criar variáveis globais

```
(function () {
    document.writeln('Olá mundo!');
})();

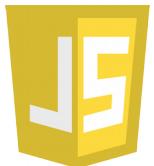
(function (nome) {
    document.writeln('Olá ' + nome + ' !');
}) ('UFAM');
```



Funções de Primeira Classe

- As funções imediatas também permitem o retorno de valores, que podem ser atribuídos a variáveis

```
var resultado = (function () {
    // fazer algum calculo mais complexo
    // com variáveis locais...
    // ...
    // return algumacoisa;
})();
```



Escopos

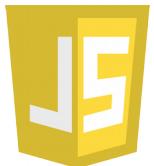
- A linguagem JavaScript suporta dois tipos de escopos: **global e local**
- Uma variável é global, isto é seu valor será acessível e modificável em todo o seu programa, quando:
 - A variável for declarada fora de uma definição de função
 - A variável for inicializada sem declará-la (com `var/let`)
- Uma variável declarada dentro de uma definição de função é local
 - Ela é criada e destruída sempre que a função é executada e não pode ser acessada por qualquer código fora da função



Escopos

- Para a grande maioria das linguagens, cada bloco de código define seu próprio escopo
- Exemplo usando a linguagem Java, onde a variável `foo` é acessível apenas dentro do bloco em que é definida:

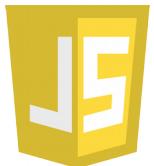
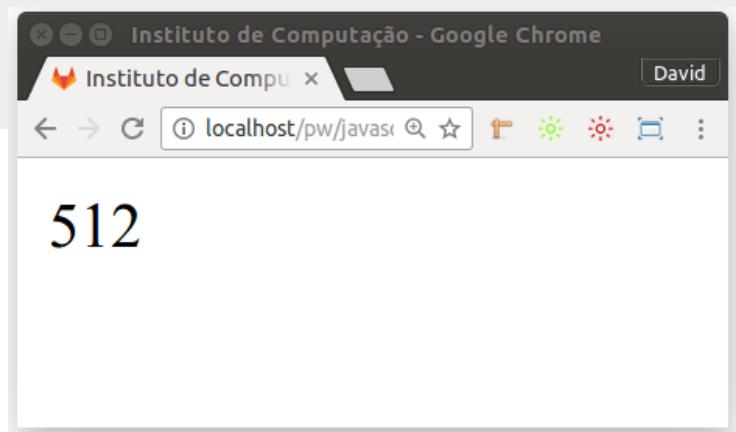
```
public static void main(String[] args) {  
    { // início de bloco  
        int foo = 4;  
    } // fim de bloco  
    System.out.println(foo); // Erro  
}
```



Escopos

- Por outro lado, para o ECMAScript 5, o escopo de uma variável é sempre a função onde ela é declarada

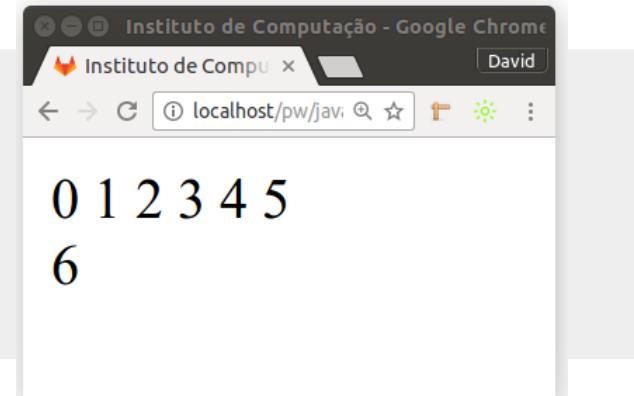
```
function foo () {  
    var x = -512;  
    if (x < 0) {  
        var tmp = -x;  
    }  
    document.writeln(tmp);  
}  
  
foo();
```



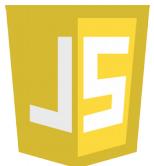
Escopos

- Desta forma, diferentemente de Java e outras linguagens, no JavaScript, uma variável declarada com **var** em um for loop acaba vazando o seu valor

```
for (var i = 0; i <= 5; i += 1) {  
    document.writeln(i);  
}  
document.writeln("<br>");  
document.writeln(i);
```



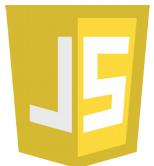
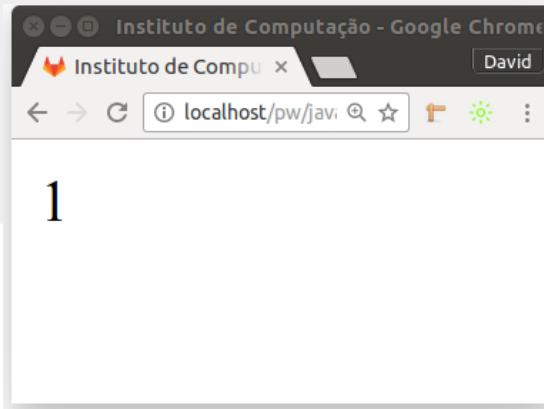
- Isso acontece porque o escopo de variável declarada com **var** é a função onde foi feita a declaração



Escopos

- Tentar acessar o valor de uma variável em um escopo onde ela não é conhecida gera um **ReferenceError**
- Tentar setar o valor de uma variável em um escopo onde ela não é conhecida cria uma nova variável global

```
function foo() {  
    a = 1; // `a` não foi declarado  
}  
  
foo();  
document.write(a);
```

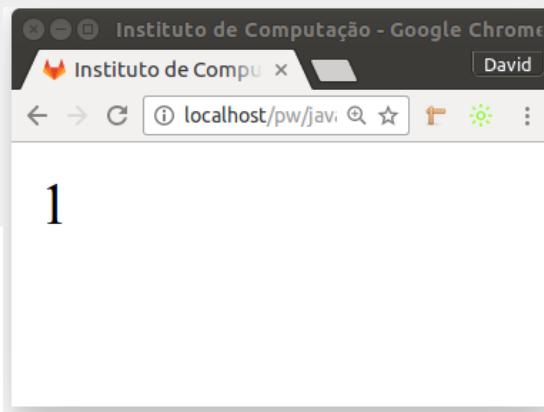


Escopos

- Tentar acessar o valor de uma variável em um escopo onde ela não é conhecida gera um **ReferenceError**
- Tentar setar o valor de uma variável em um escopo

O uso de variáveis globais em JavaScript é uma prática muito ruim.
Evite ao máximo essa prática! Sempre declare suas variáveis!

```
function foo() {  
    a = 1; // `a` não foi declarado  
}  
  
foo();  
document.write(a);
```



Escopos

- A declaração **let** chega na ECMAScript 6 (de 2015) como um substituto para a declaração **var**
 - A ideia é que **var** seja descontinuado em um futuro distante, pois hoje seria impossível não suportá-lo sem quebrar 100% da internet
- O **let** funciona como o esperado, definindo a variável no local onde ela foi declarada

```
function foo() {
    let a = true;
    if (a) {
        let b = 'ufam';
        document.writeln(typeof b);
    }
    document.writeln(typeof b);
}
foo();
```



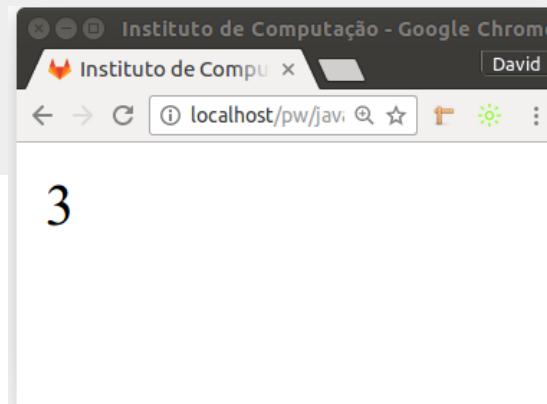
Escopos e Hoisting

- Sempre que um **var** ou **function** aparecer dentro de um escopo, essa declaração é visível em qualquer lugar desse escopo

```
foo();  
function foo () {  
    a = 3;  
    var a;  
    document.writeln( a );  
}
```

Funciona porque **foo()** é visivel em qualquer lugar do escopo (hoisting)

Declaração movida para o início de **foo()**



Exercício

- Qual é o resultado da execução do seguinte código?
Por que ele teve a saída informada?

```
function teste() {  
  
    console.log(a);  
    console.log(foo());  
  
    var a = 1;  
    function foo() {  
        return 2;  
    }  
}  
  
teste();
```



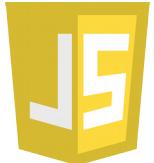
Cadeias de Escopo

- Em JavaScript, uma função tem acesso a todas as variáveis presentes nos escopos de seus pais

```
var global = 1;
function outer() {
    var outer_local = 2;
    function inner() {
        var inner_local = 3;
        return inner_local + outer_local + global;
    }
    return inner();
}
document.write(outer());
```



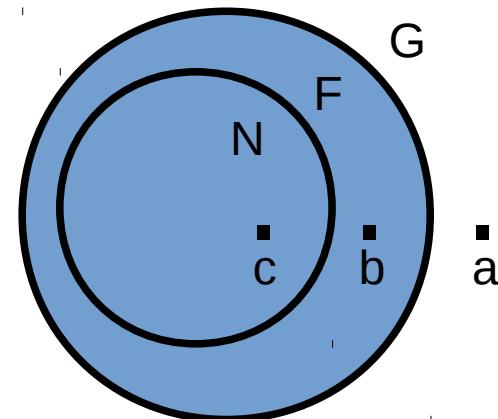
- Isso possibilita a existência de **cadeias de escopo**, que podem ser tão profundas quanto desejado



Closures

- Closure é um dos recursos mais fundamentais de JavaScript, mas comumente é mal compreendido
- Para compreender closures, considere a seguinte cadeia de escopos

```
var a = "global variable";
var F = function () {
    var b = "local variable";
    var N = function () {
        var c = "inner local";
    };
};
```

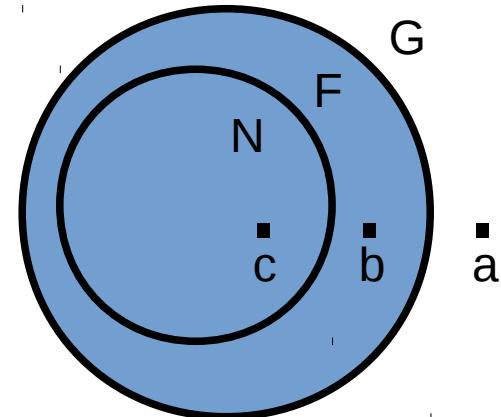


- O que aconteceria se pudéssemos chamar a função **N()** a partir do escopo global?

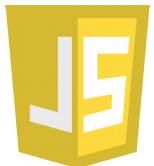


Closures

```
var a = "global variable";
var F = function () {
    var b = "local variable";
    var N = function () {
        var c = "inner local";
    };
};
}
```

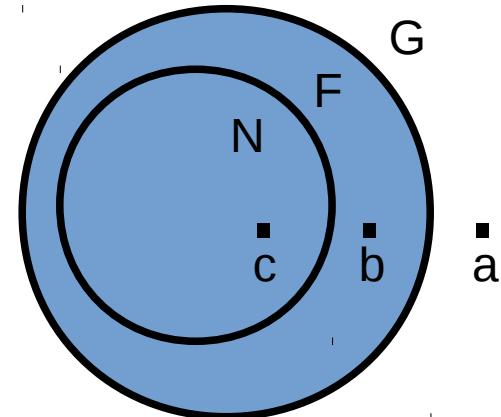


- O que aconteceria se pudéssemos chamar a função `N()` a partir do escopo global?
 - Como a variável lembra de seu ambiente, `N()` continuaria com acesso ao escopo de `F()`
 - Isto é, a função `N()` seria a única forma de acesso da variável `b` dentro do escopo de `G`
 - Note que, neste caso, a variável `b` funcionaria como uma espécie de variável privada de `N()`



Closures

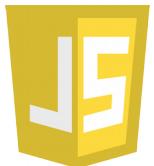
```
var a = "global variable";
var F = function () {
    var b = "local variable";
    var N = function () {
        var c = "inner local";
    };
};
}
```



- Essa quebra da cadeia de escopos e geração de um espaço privado para a função é chamado de **closure**

com acesso ao escopo de **F()**

- Isto é, a função **N()** seria a única forma de acesso da variável **b** dentro do escopo de **G**
- Note que, neste caso, a variável **b** funcionaria como uma espécie de variável privada de **N()**

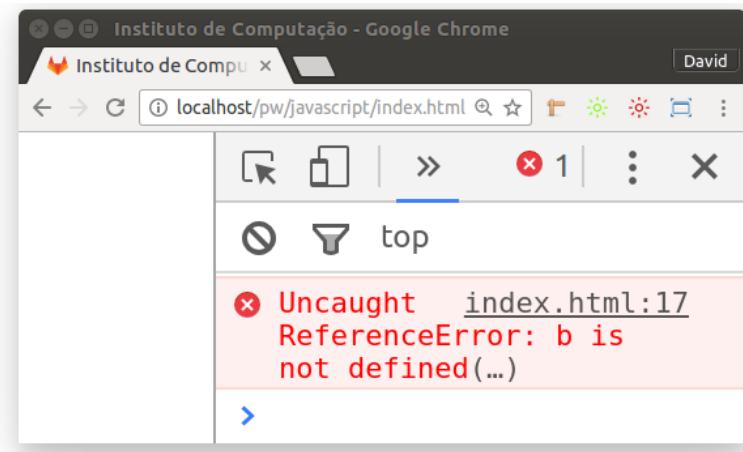


Closures

- Considere o código abaixo, que é similar ao anterior, mas com `F()` retornando `N()`, e `N()` retornando `b`
- Note que não temos acesso à variável `b` a partir do escopo global

```
var a = "global variable";
var F = function () {
    var b = "local variable";
    var N = function () {
        var c = "inner local";
        return b;
    };
    return N;
};

document.write(b);
```

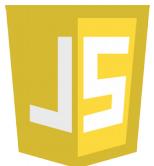


Closures

- No entanto, podemos fazer uma variável de escopo global receber **F()**, tal como mostra o código abaixo
- O resultado: uma nova função global que acessa o espaço privado de **F()**, formando um **closure**

```
var a = "global variable";
var F = function () {
    var b = "local variable";
    var N = function () {
        var c = "inner local";
        return b;
    };
    return N;
};

var inner = F();
document.write(inner());
```



Closures

- O código a seguir mostra um exemplo típico de aplicação de closures

```
function criarIncremento(inicio) {  
    return function () {  
        inicio++;  
        return inicio;  
    }  
}  
  
var inc = criarIncremento(5);  
  
var a = inc();  
document.writeln(a);  
var b = inc();  
document.writeln(b);  
var c = inc();  
document.writeln(c);
```



Exercício

- O que faz o código abaixo? Ele usa recurso de closure?

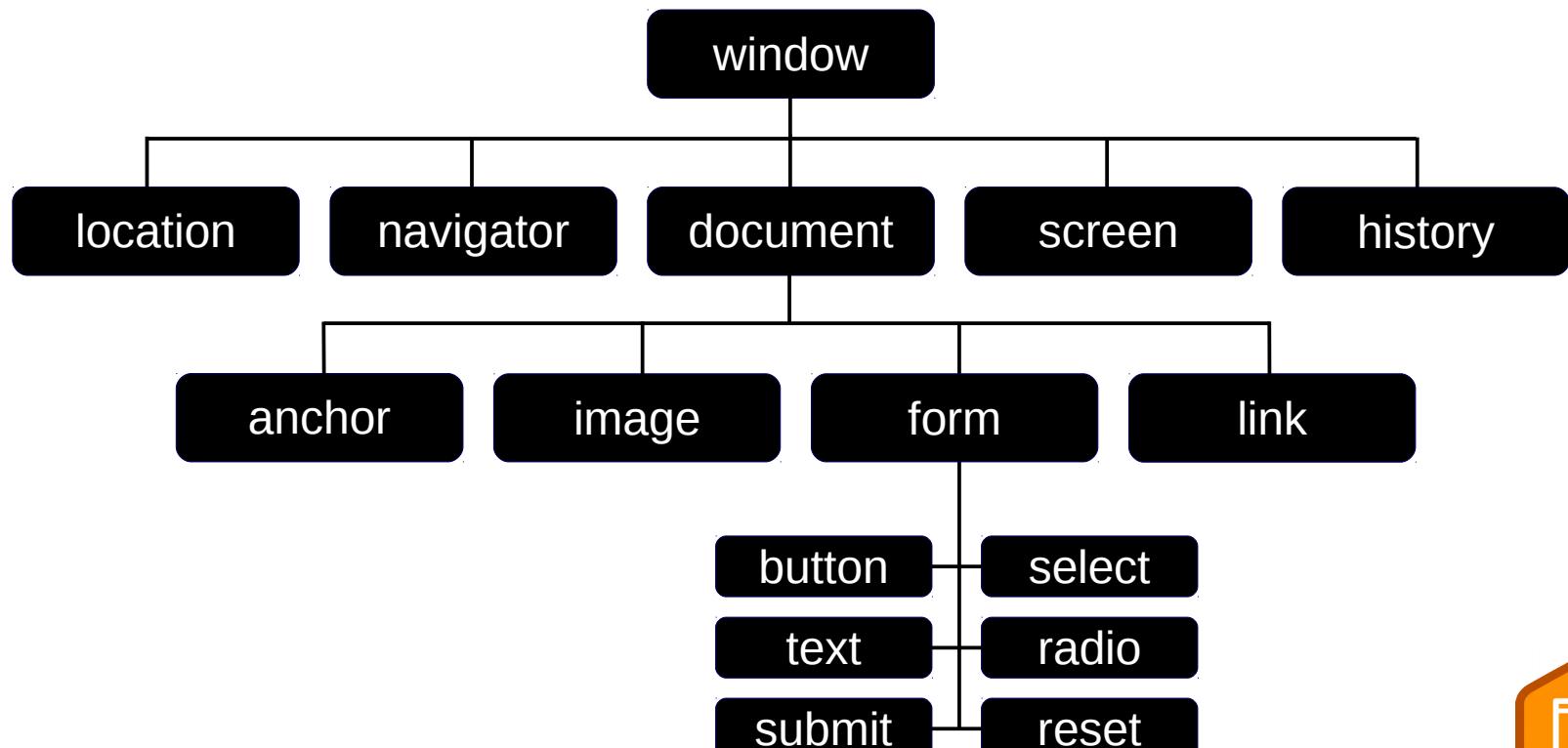
```
var getValue, setValue;
(function () {
    var secret = 0;
    getValue = function () {
        return secret;
    };
    setValue = function (v) {
        if (typeof v === "number") {
            secret = v;
        }
    };
}());

document.writeln(getValue());
setValue(8);
document.writeln(getValue());
setValue(false);
document.writeln(getValue());
```



Trabalhando com a Dom

- Quando um browser carrega uma página, ele cria uma hierarquia de objetos para representar essa página

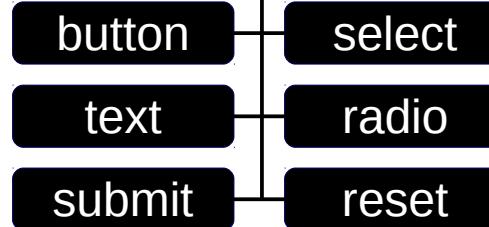


Trabalhando com a Dom

- Quando um browser carrega uma página, ele cria uma hierarquia de objetos para representar essa página

Essa hierarquia de objetos é chamada **DOM (Document Object Model)** e ela fica armazenada em memória

A DOM possui uma ligação viva com o conteúdo página, de forma que mudanças neste modelo são imediatamente refletidas na página Web



Objeto **window**

- O objeto **window** representa uma aba do browser, e é o nó raiz do Document Object Model (DOM)
- Tecnicamente, toda variável ou função de escopo global pertence ao objeto window
 - Desta forma, as duas sequências de códigos abaixo são similares:

```
document.write("a test message");
alert("Hello");
foo = "bar";
```

```
window.document.write("a test message");
window.alert("Hello");
window.foo = "bar";
```



Objeto `window`

- Algumas propriedades e métodos do objeto `window`:
 - `alert()`, `prompt()`, `confirm()`
 - `open()` e `close()`
 - `setTimeout()` e `setInterval()`
 - `navigator`, `document` e `screen`
 - `location` e `history`

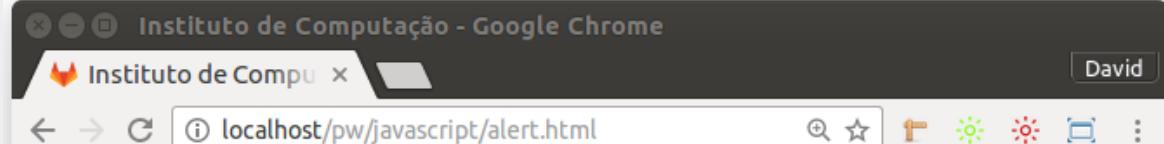


Objeto **window**

- Algumas proriedades e métodos do objeto **window**:
 - **alert()**, **prompt()**, **confirm()**

```
– <pre>document.writeln(window.navigator.userAgent);</pre>
```

```
– <pre>`</pre>
```



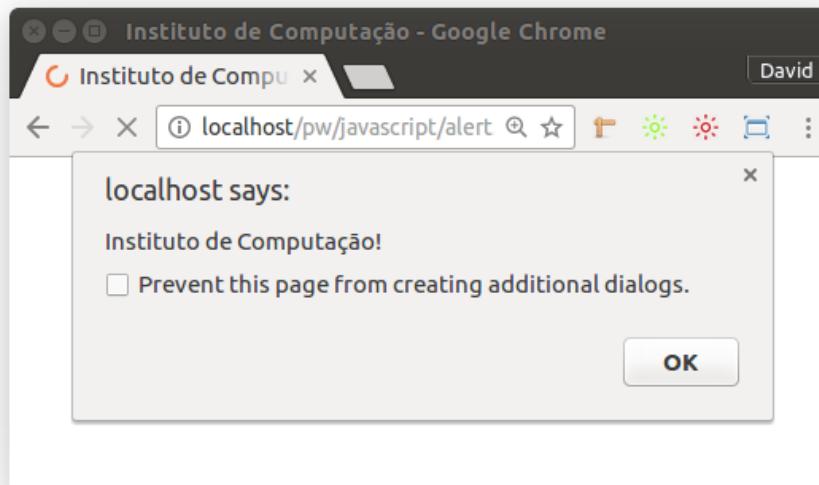
Alert, Confirm e Prompt

- Interação com usuários usando caixas de diálogo

window.alert(msg);

- Mostra uma mensagem em uma caixa que é fechada quando o usuário clica no botão OK

```
window.alert("Instituto de Computação!")
```

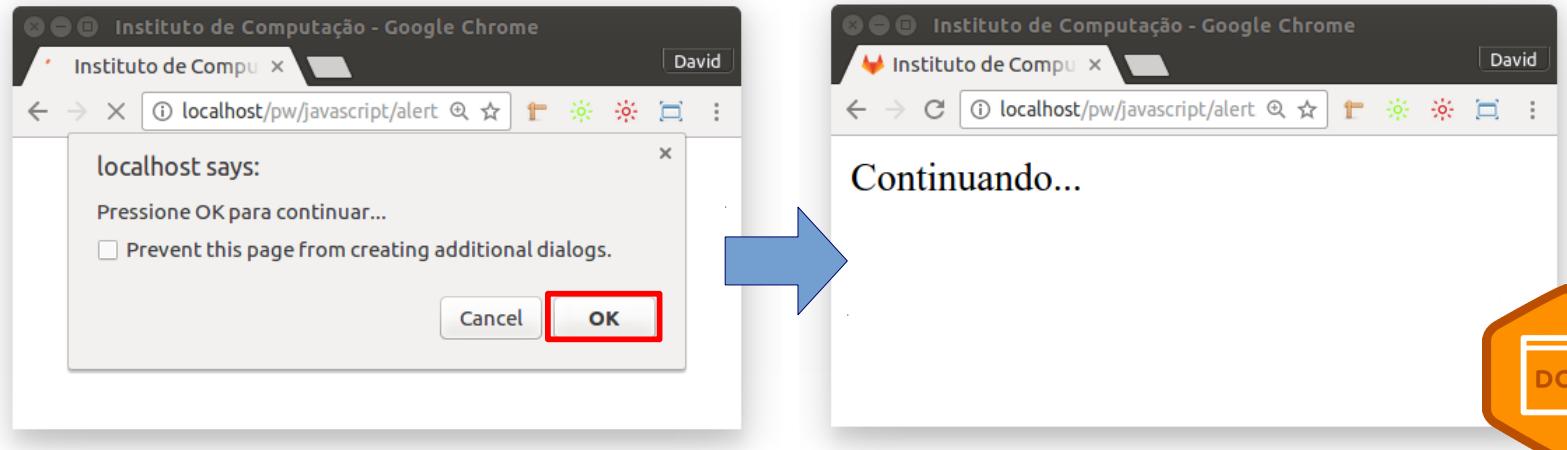


Alert, Confirm e Prompt

```
var r = window.confirm(msg);
```

- Mostra uma mensagem junto com os botões OK e Cancel.
- Retorna **true** se o usuário clica em OK, e **false** caso contrário

```
r = window.confirm("Pressione OK para continuar...");  
if (r) document.writeln("Continuando...");
```

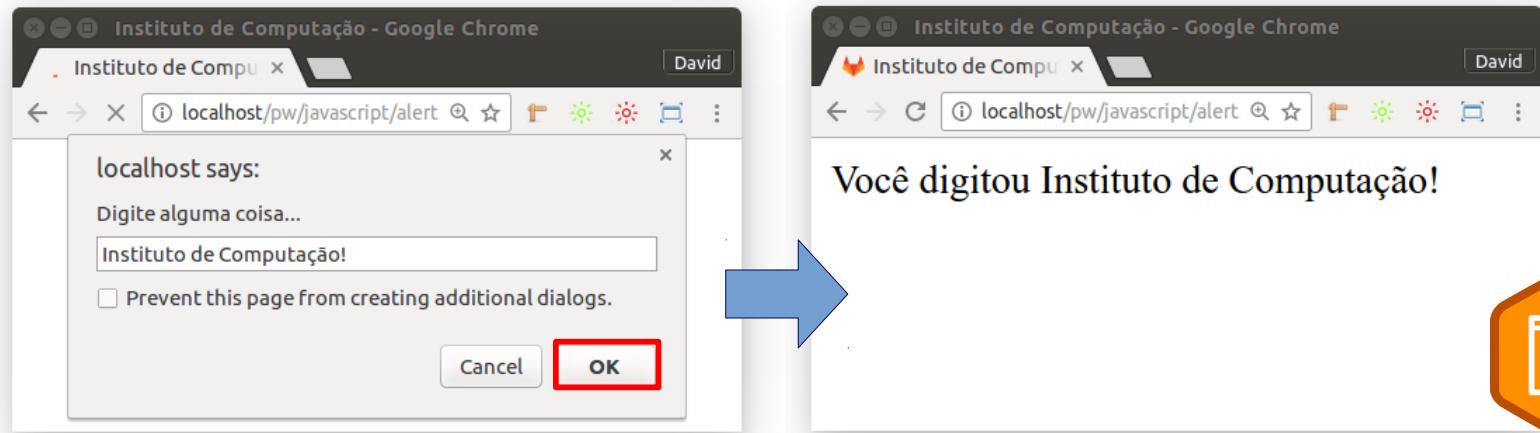


Alert, Confirm e Prompt

```
var r = window.prompt(msg, default);
```

- Mostra uma mensagem e um input de dados
- O método retorna o dado informado pelo usuário, ou `null` caso o botão **Cancel** seja clicado

```
dado = window.prompt("Digite alguma coisa...");  
if (dado) document.writeln("Você digitou " + dado);
```



Objeto **document**

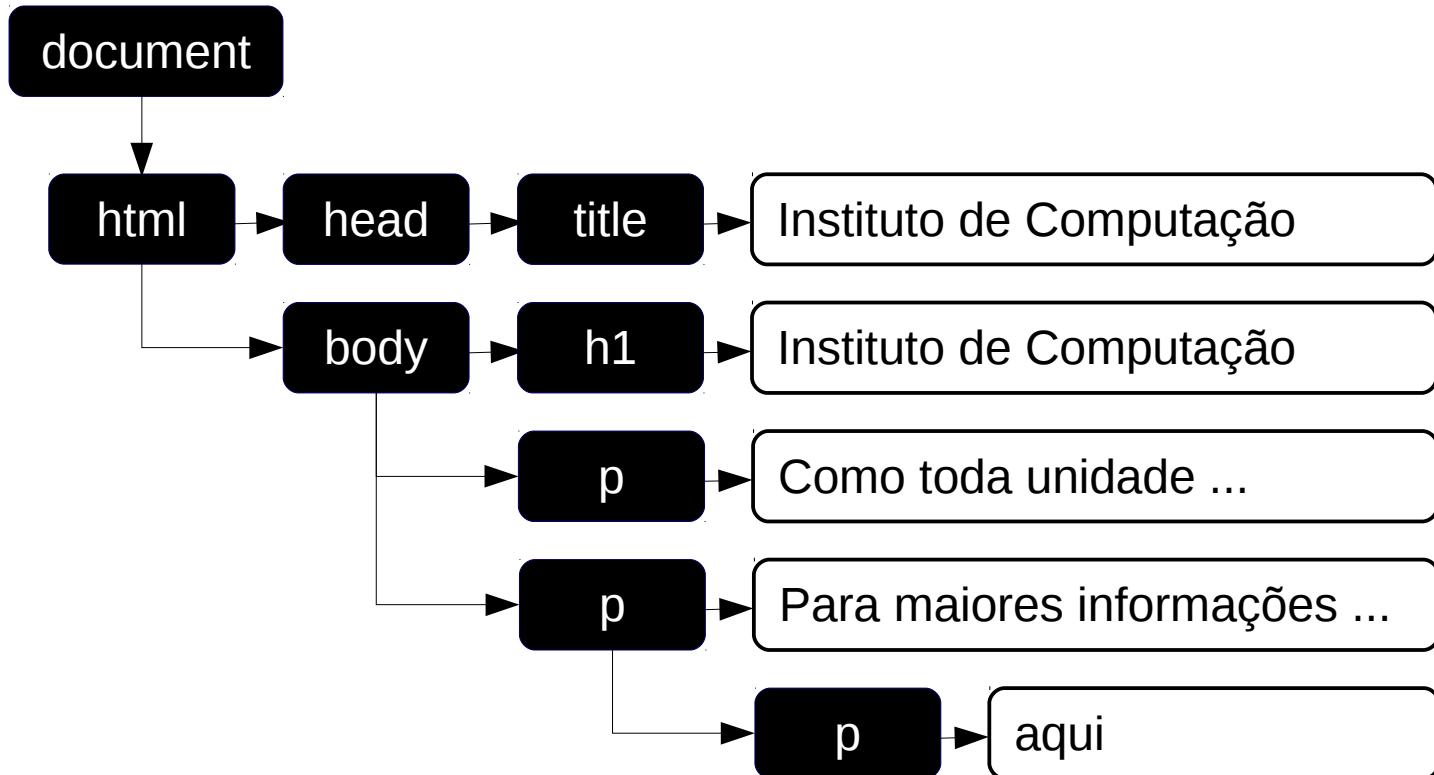
- O objeto **document** representa a página carregada em uma aba, e é o ponto de acesso aos elementos da página

```
<html>
  <head>
    <title>Instituto de Computação</title>
  </head>
  <body>
    <h1>Instituto de Computação</h1>
    <p>Como toda unidade acadêmica, o Instituto de
       Computação atua no ensino, pesquisa e extensão.</p>
    <p>Para mais informações sobre o Icomp, clique
      <a href="http://icomp.ufam.edu.br">Aqui</a>
    <p>
  </body>
</html>
```



Objeto `document`

- O objeto **document** representa a página carregada em uma aba, e é o ponto de acesso aos elementos da página



Objeto document

- O objeto **document** também provê alguns arrays de objetos específicos da página carregada, tais como:
 - **images []**, **forms []**, **links []**

```
<body>
  <h1>Instituto de Computação</h1>

  <p>Como toda unidade acadêmica,
  o Instituto de Computação atua no
  ensino, pesquisa e extensão.</p>

  <p>Para mais informações sobre o Icomp, clique
    <a href="http://icomp.ufam.edu.br">Aqui</a>
  <p>

    <script type="text/javascript">
      alert(document.links[0].href);
    </script>

  </body>
```

Primeiro link da
página: **links [0]**



Objeto document

- O objeto **document** também provê alguns arrays de objetos específicos da página carregada, tais como:
 - images []**, **forms []**, **links []**

The screenshot shows a Google Chrome window with the title "Instituto de Computação - Google Chrome". The address bar shows "localhost/pw/javascript/alert.html". A JavaScript alert dialog is displayed with the message "localhost says: http://icomp.ufam.edu.br/" and an "OK" button. The main content area of the browser shows the following HTML code:

```
<body>
  <h1>Instituto de Compu</h1>
  <p>Como toda universidade, o Instituto de Ciências de Ensino, pesquisa e extensão.</p>
  <p>Para mais informações sobre o Icomp, clique <a href="http://icomp.ufam.edu.br">Aqui</a></p>
  <script type="text/javascript">
    alert(document.links[0].href);
  </script>
</body>
```

A blue callout bubble points to the first link in the page's script: "Primeiro link da página: links[0]".



Objeto **document**

- Um dos usos do objeto **document** é prover o programador com informações sobre o documento

Property	Description	Returns
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

Objeto **document**

- Um dos usos do objeto **document** é prover o programador com informações 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

Objeto **document**

- Um dos usos do objeto **document** é prover o programador com informações sobre o documento

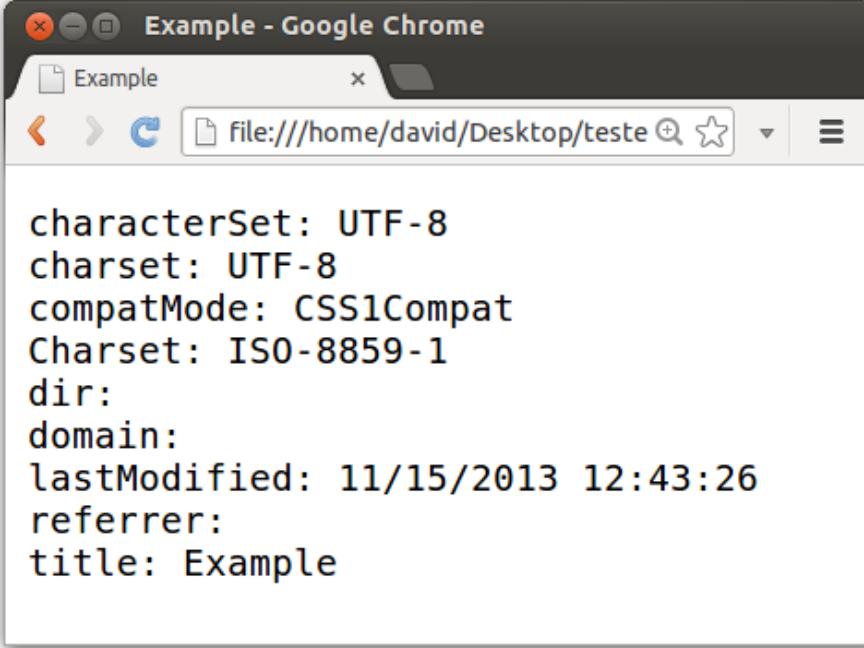
```
<script>
    document.writeln("<pre>");
    document.writeln("characterSet: " + document.characterSet);
    document.writeln("charset: " + document.charset);
    document.writeln("compatMode: " + document.compatMode);
    document.writeln("Charset: " + document.defaultCharset);
    document.writeln("dir: " + 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>
```



Objeto document

- Um dos usos do objeto **document** é prover o programador com informações sobre o documento

```
<script>
  document.writeln("<pre>");
  document.writeln("characterSet: " + document.characterSet);
  document.writeln("char");
  document.writeln("com");
  document.writeln("Cha");
  document.writeln("dir");
  document.writeln("dom");
  document.writeln("las");
  document.writeln("ref");
  document.writeln("tit");
  document.write("</pre>");
</script>
```



```
characterSet: UTF-8
charset: UTF-8
compatMode: CSS1Compat
Charset: ISO-8859-1
dir:
domain:
lastModified: 11/15/2013 12:43:26
referrer:
title: Example
```



Recuperando Nós

- Ao se trabalhar com a DOM, frequentemente precisamos buscar e acessar os objetos da página Web
 - Após acessar um objeto, podemos ler ou editar o conteúdo e as propriedades de seu elemento
- Essa capacidade é provida por um dado conjunto de métodos do objeto **document**, dos quais destacamos:
 - `getElementById()`
 - `getElementsByTagName()`
 - `getElementsByName()`
 - `getElementsByClassName()`



Recuperando Nós

```
<h1>Instituto de Computação</h1>  
  
<p>Como toda unidade acadêmica, o Instituto de  
Computação atua no ensino, pesquisa e extensão.</p>  
  
  
  
  
  
<pre id="results"></pre>
```

```
var results = document.getElementById("results");  
  
var pElems = document.getElementsByTagName("p");  
results.innerHTML += "\nQtd parágrafos: " + pElems.length;  
  
var fruitsElems = document.getElementsByClassName("fruta");  
results.innerHTML += "\nQtd frutas: " + fruitsElems.length;  
  
var nameElems = document.getElementsByName("maca");  
results.innerHTML += "\nQtd maçãs: " + nameElems.length;
```



Recuperando Nós

```
<h1>Instituto de Computação</h1>  
  
<p>Como toda unidade acadêmica, o Instituto de  
Computação atua no ensino, pesquisa e extensão.</p>  
  
  
  
  
  
<pre id="results"></pre>  
  
var results = document.getElementById("results");  
  
var pElems = document.getElementsByTagName("p");  
results.innerHTML += "\nQtd parágrafos: " + pElems.length;  
  
var fruitsElems = document.getElementsByClassName("fruta");  
results.innerHTML += "\nQtd frutas: " + fruitsElems.length;  
  
var nameElems = document.getElementsByName("maca");  
results.innerHTML += "\nQtd maçãs: " + nameElems.length;
```

Podemos usar a propriedade **innerHTML** para ler ou alterar o conteúdo de um elemento



Recuperando Nós

```
<h1>Instituto de Computação</h1>
```

```
<p>Como toda unidade acadêmica, o Instituto de Computação atua no ensino, pesquisa e extensão.
```

```
<img id="limao" class="fruta" name="limao" alt="Lemon" />
<img id="maca" class="fruta" name="maca" alt="Apple" />
<img id="banana" class="fruta" name="banana" alt="Banana" />
<pre id="results"></pre>
```

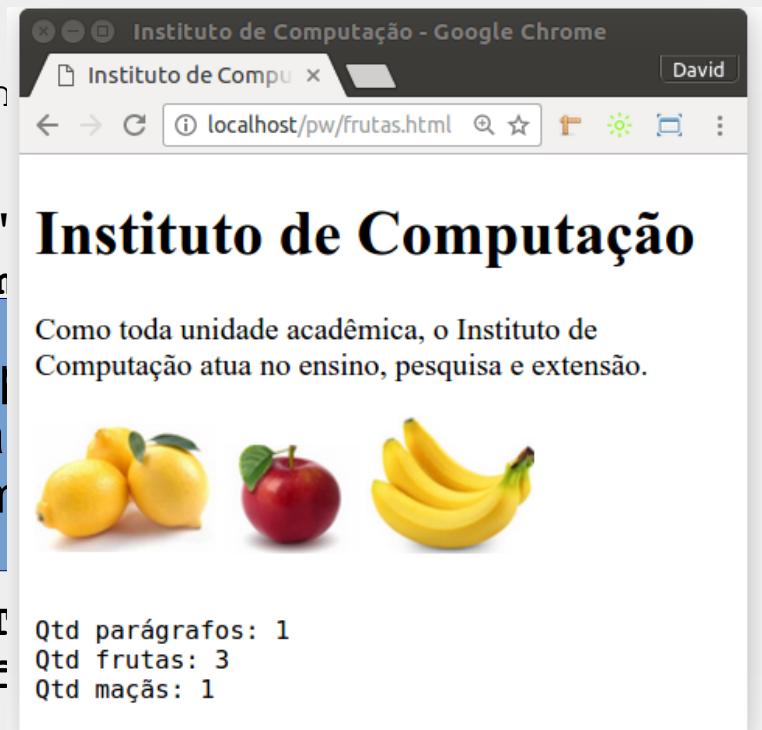
```
var results = document.getElementById("results");
```

```
var pElems = document.getElementsByTagName("p");
results.innerHTML += "\nQtd parágrafos: " + pElems.length;
```

```
var fruitsElems = document.getElementsByClassName("fruta"),
results.innerHTML += "\nQtd frutas: " + fruitsElems.length;
```

```
var nameElems = document.getElementsByName("maca");
results.innerHTML += "\nQtd maçãs: " + nameElems.length;
```

Podemos usar a propriedade `innerHTML` para recuperar o conteúdo de um elemento.



Recuperando Nós

- Através das funções abaixo, podemos recuperar elementos usando seletores CSS:
 - `querySelector()`
 - `querySelectorAll()`
- No exemplo abaixo, usamos um seletor para selecionar todos os elementos `p` e a imagem com `id=maca`

```
<script>
    var results = document.getElementById("results");
    var elems = document.querySelectorAll("p, img#maca");

    results.innerHTML = "Qtd elementos: " + elems.length;
</script>
```



Recuperando Nós

- Através das funções abaixo, podemos recuperar elementos usando seletores CSS
- `querySelector()`
- `querySelectorAll()`
- No exemplo abaixo, usamos o seletor All para todos os elementos `p` e a imagem de uma maçã

```
<script>
    var results = document.getElementById("results");
    var elems = document.querySelectorAll("p, img#maca");

    results.innerHTML = "Qtd elementos: " + elems.length;
</script>
```



Recuperando Nós

- Imagens e itens de formulários também podem ser selecionados através da propriedade **name**
- Objetos com a propriedade **name** podem ser referenciados diretamente através do objeto **document**

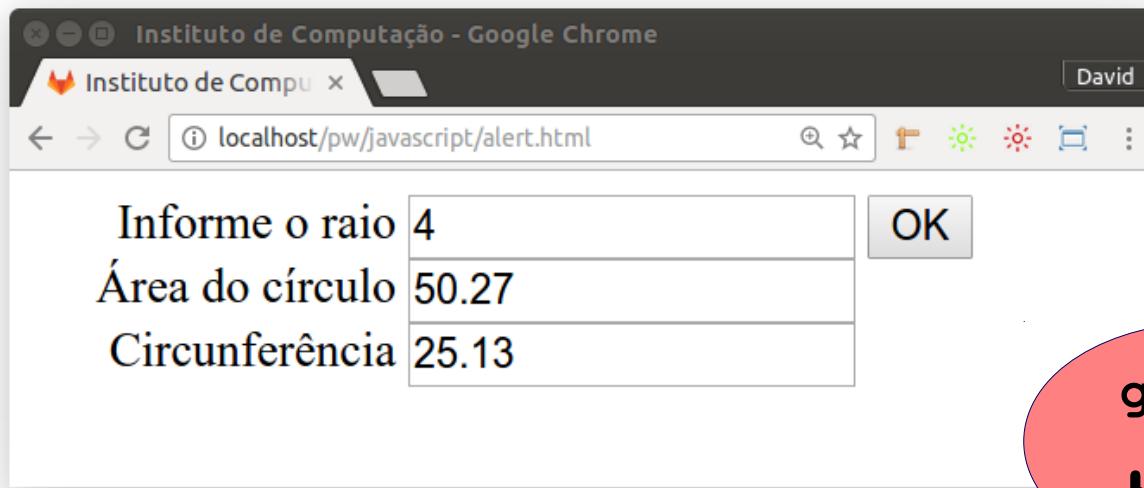
```
<form name="myForm">
    <input name="myInput" type="text" value="">
</form>
```

- No exemplo acima, o JavaScript pode referenciar o formulário usando **document.myForm**
- Da mesma forma, o elemento input pode ser selecionado através de **document.myForm.myInput**



Exercício

- Escreva uma página similar à apresentada abaixo, que calcula a área e a circunferência de um círculo cujo raio é informado pelo usuário:



github
JS2



Trabalhando com Classes

- Através da propriedade `className`, pode-se recuperar ou editar a lista de 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>
```

```
botao = document.getElementById("pressme");
botao.onclick = function(e) {
    paragrafo = document.getElementById("textblock");
    paragrafo.className += " newclass";
};
```



Trabalhando com Classes

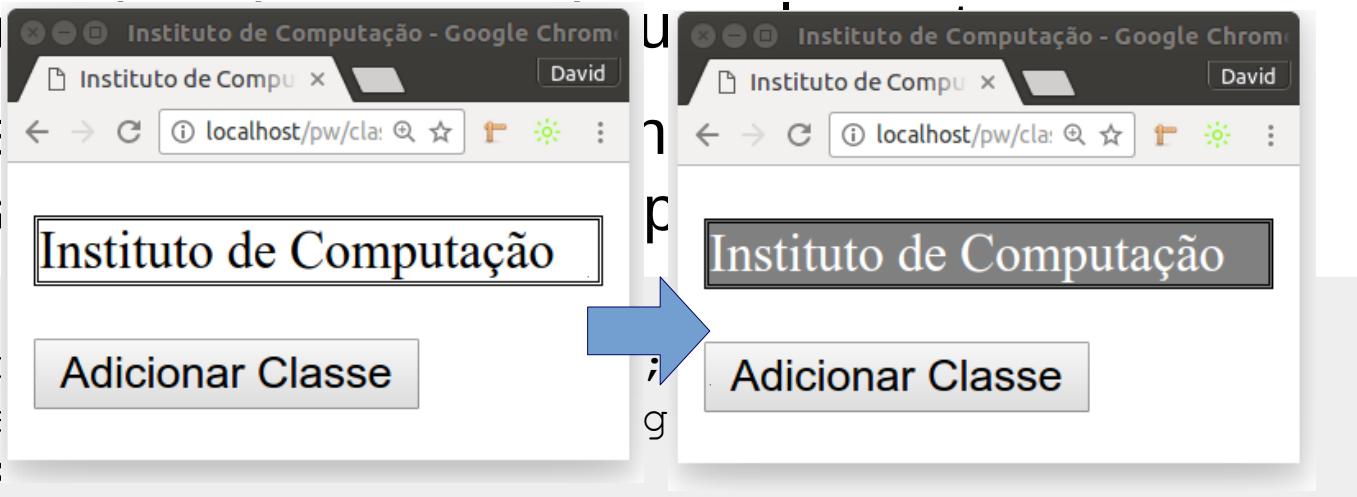
- Através da propriedade **className**, pode-se recuperar ou editar.

- Pode adicionar ou mudar.

```
<style>  
p { border: 1px solid black; }  
p.newclass { background-color: #ccc; }  
</style>
```

```
<p id="textblock">Instituto de Computação</p>  
<button id="pressme">Adicionar Classe</button>
```

```
botao = document.getElementById("pressme");  
botao.onclick = function(e) {  
    paragrafo = document.getElementById("textblock");  
    paragrafo.className += " newclass";  
};
```



Trabalhando com StyleSheets

- Para obter os estilos de um determinado elemento, usamos a propriedade **style**
 - 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");
var botao = document.getElementById("pressme");
botao.onclick = function() {
    icomp.style.cssText = "color:black";
}
```



Trabalhando com StyleSheets

- 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



Trabalhando com StyleSheets

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

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

```
var icomp = document.getElementById("icomp");
var botao = document.getElementById("pressme");
botao.onclick = function() {
    icomp.style.fontSize = "32px";
    icomp.style.color = "red";
}
```

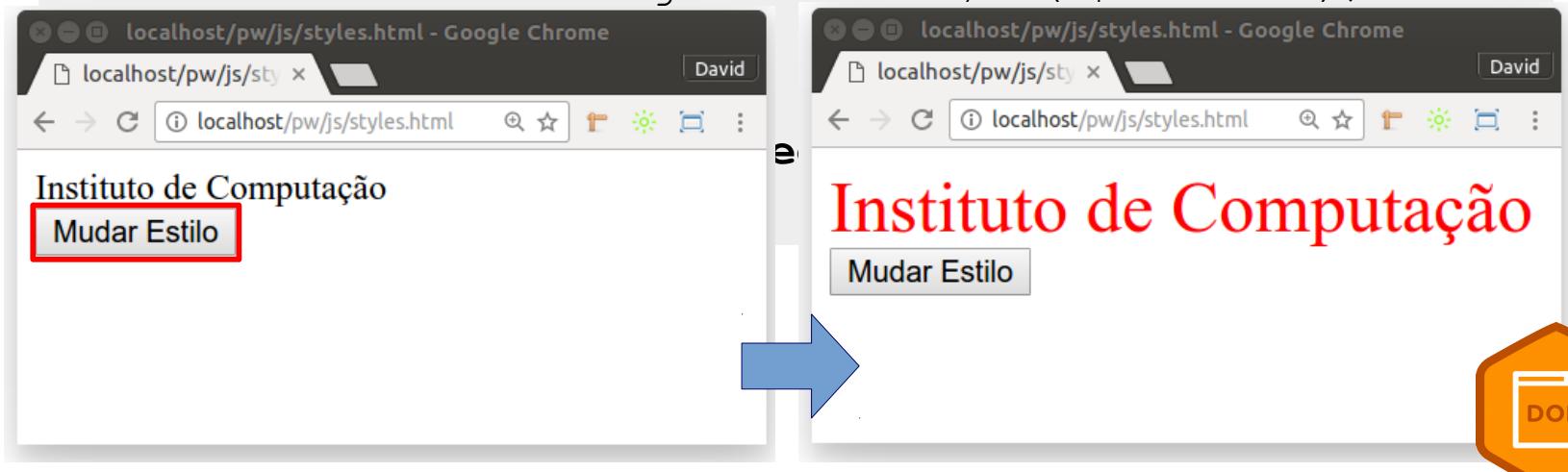


Trabalhando com StyleSheets

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

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

```
var icomp = document.getElementById("icomp");
var botao = document.getElementById("pressme");
```



Trabalhando com StyleSheets

- Propriedades CSS e suas equivalências na linguagem JavaScript:

- **background-color**
- **border-radius**
- **font-size**
- **list-style-type**
- **word-spacing**
- **z-index**
- **backgroundColor**
- **borderRadius**
- **fontSize**
- **listStyleType**
- **wordSpacing**
- **zIndex**



Trabalhando com StyleSheets

- 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>
```

```
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");
}
```

Trabalhando com StyleSheets

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



Trabalhando com Atributos

- Após a seleção de um dado elemento, também é possível ler ou editar qualquer atributo desse 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

- Após a seleção de um dado elemento, também é possível ler ou editar qualquer atributo desse elemento

```
<a href="#" id="troca-logo">Trocar Logo</a>


var ufamImg = new Image();
var icompImg = new Image();

ufamImg.setAttribute('src', 'ufam.png');
icompImg.setAttribute('src', 'icomp.png');

troca = document.getElementById("troca-logo");

troca.onclick = function () {
    var logo = document.logo;
    if (logo.getAttribute('src') == 'ufam.png') {
        logo.setAttribute('src', 'icomp.png');
    } else {
        logo.setAttribute('src', 'ufam.png');
    }
}
```

Trabalhando com Atributos

- Após a seleção de um dado elemento, também é possível ler ou editar qualquer atributo desse elemento

```
<a href="#" id="troca-logo">Trocar Logo</a>

```

```
var ufamTma = new Image();
var ic
ufamIn
icomp]
troca
troca.
va:
if
}
{
}
}
```



Exercício

- Escreva um programa que gere o gráfico de barras da figura abaixo a partir dos valores informados



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

Modificando o Modelo

- Funções de manipulação da árvore DOM

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

Modificando o Modelo

- Funções de manipulação da árvore DOM

Member	Description	Returns
isSameNode(HTMLElement)	Determines if the specified element is the same as the current element	boolean
outerHTML	Gets or sets an element's HTML and contents	string
removeChild(HTMLElement)	Removes the specified child of the current element	HTMLElement
replaceChild(HTMLElement, HTMLElement)	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(<tag>)</code>	Creates a new HTMLElement object with the specific tag type	HTMLElement
<code>createTextNode(<text>)</code>	Creates a new Text object with the specified content	Text

Modificando o Modelo

```
<table id="tabelaFrutas">
  <tr><th>Nome</td><th>Cor</td></tr>
  <tr><td>Banana</td><td>Amarelo</td></tr>
  <tr><td>Apple</td><td>Vermelho/Verde</td></tr>
</table>
<button id="add">Adicionar</button>
<button id="rem">Remover</button>
```

```
var tabela = document.getElementById("tabelaFrutas");
document.getElementById("add").onclick = function() {
  var row = tabela.appendChild(document.createElement("tr"));
  row.setAttribute("id", "newrow");
  row.appendChild(document.createElement("td"))
    .appendChild(document.createTextNode("Morango"));
  row.appendChild(document.createElement("td"))
    .appendChild(document.createTextNode("Vermelho"));
};

document.getElementById("rem").onclick = function() {
  var row = document.getElementById("newrow");
  row.parentNode.removeChild(row);
}
```

Modificando o Modelo

```
<table id="tabelaFrutas">
  <tr><th>Nome</th><th>Cor</th></tr>
  <tr><td>Banana</td><td>Amarelo</td></tr>
  <tr><td>Maçã</td><td>Vermelho/Verde</td></tr>
</table>

<button type="button" onclick="adicionar();">Adicionar</button>
<button type="button" onclick="remover();">Remover</button>

var tabela = document.getElementById("tabelaFrutas");
var row = document.createElement("tr");
var nome = document.createElement("td");
var cor = document.createElement("td");
var rem = document.createElement("button");
var add = document.createElement("button");

function adicionar() {
  nome.innerHTML = prompt("Digite o nome da fruta");
  cor.innerHTML = prompt("Digite a cor da fruta");
  row.appendChild(nome);
  row.appendChild(cor);
  tabela.appendChild(row);
}

function remover() {
  var row = document.getElementById("newrow");
  row.parentNode.removeChild(row);
}

// Exemplo de uso
adicionar();
adicionar();
adicionar();

// Removendo uma linha
var row = document.getElementById("newrow");
row.parentNode.removeChild(row);
```

Modificando o Modelo

- 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

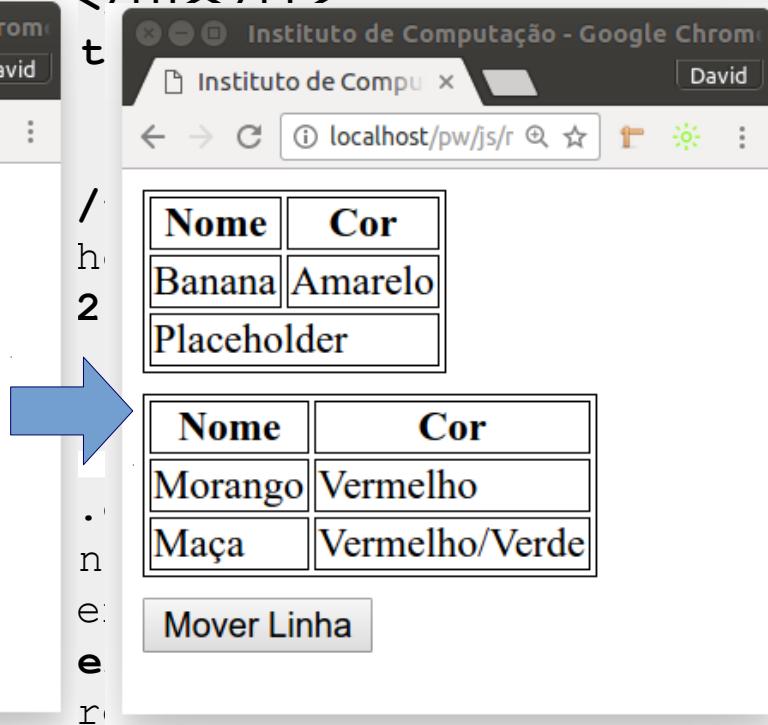
Modificando o Modelo

```
<table>
  <tr><th>Nome</th><th>Cor</th></tr>
  <tr><td>Banana</td><td>Amarelo</td></tr>
  <tr id="origem"><td>Maça</td><td>Vermelho/Verde</td></tr>
</table>
<table>
  <tr><th>Nome</th><th>Cor</th></tr>
  <tr><td>Morango</td><td>Vermelho</td></tr>
  <tr id="destino"><td colspan="2">Placeholder</td></tr>
</table>
<button id="mover">Mover Linha</button>
```

```
document.getElementById("mover").onclick = function() {
  var origem = document.getElementById("origem");
  var destino = document.getElementById("destino");
  destino.innerHTML = origem.innerHTML;
  origem.outerHTML = '<tr id="target"><td colspan="2">' +
    'Placeholder</td>';
};
```

Modificando o Modelo

```
<table>
  <tr><th>Nome</th><th>Cor</th></tr>
  <tr><td>Banana</td><td>Amarelo</td></tr>
  <tr>
    <td>Maça</td>
    <td>Vermelho/Verde</td>
  </tr>
</table>
<table>
  <tr><th>Nome</th><th>Cor</th></tr>
  <tr><td>Banana</td><td>Amarelo</td></tr>
  <tr>
    <td>Placeholder</td>
    <td>Placeholder</td>
  </tr>
</table>
<button>Mover Linha</button>
document.addEventListener('click', function(event) {
  var target = event.target;
  var rows = document.querySelectorAll('table tr');
  if (target === document.querySelector('button')) {
    var moveRow = rows[2];
    if (moveRow) {
      moveRow.classList.add('highlight');
      var nextRow = rows[3];
      if (nextRow) {
        moveRow.insertBefore(nextRow);
        moveRow.classList.remove('highlight');
      }
    }
  }
});
```



Trabalhando com Eventos

- Em algumas interfaces, a única forma de identificar se uma dada tecla foi ou não pressionada é ler constantemente o estado daquela tecla
 - Esta abordagem é chamada **pooling**, e deve ser evitada tanto quanto possível
- No entanto, os browsers permitem a definição de manipuladores de eventos, que reagem à determinados tipos de eventos assim que eles ocorrem

```
window.addEventListener("click", function() {  
    console.log("Você clicou na página!");  
}) ;
```

Trabalhando com Eventos

- Cada manipulador de evento é registrado em um dado contexto, tal como **window** no exemplo anterior
- Cada elemento da árvore DOM possui seu próprio método **addEventListener**
 - Desta forma, todos os elementos podem ser “ouvidos”

```
<button>Me clique!</button>
<script>
    var button = document.querySelector("button");
    button.addEventListener("click", function() {
        console.log("Botão clicado!");
    });
</script>
```

Trabalhando com Eventos

- Cada manipulador de evento é registrado em um dado contexto, tal como **window** no exemplo anterior
- Também é possível manipular cliques com o método **onclick**, também presente nos elementos DOM.
No entanto, só é possível registrar um evento onclick por elemento.
O método **addEventListener** permite adicionar vários manipuladores para um determinado tipo de evento.

```
var button = document.querySelector('button')  
button.addEventListener("click", function() {  
    console.log("Botão clicado!");  
});  
</script>
```

Trabalhando com Eventos

- Podemos usar o método `removeEventListener` para apagar o registro de um dado manipulador de evento

```
<button>Clique me!</button>
<script>
  var button = document.querySelector("button");
  function once() {
    console.log("Done.");
    button.removeEventListener("click", once);
  }
  button.addEventListener("click", once);
</script>
```

Objetos de Eventos

- As funções de manipulação de eventos recebem como parâmetro um objeto com dados sobre o evento ocorrido
- Por exemplo, para saber qual botão foi clicado em um evento **mousedown**, podemos recorrer a esse objeto

```
<button>Me clique!</button>
<script>
    var button = document.querySelector("button");
    button.addEventListener("mousedown", function(event) {
        if (event.which == 1)
            console.log("Botão esquerdo");
        else if (event.which == 3)
            console.log("Botão direito");
    });
</script>
```

Propagação de Eventos

- Manipuladores de eventos de um dado nó também irão receber eventos que ocorrem nos filhos desse nó
 - Se um botão dentro de um `div` é clicado, o elemento `div` também irá receber o evento click
- No entanto, se o `div` e o botão tiverem manipuladores de eventos, o manipulador do elemento mais específico – o botão – será executado primeiro

Propagação de Eventos

- Em qualquer momento, um manipulador de eventos pode chamar o método **stopPropagation()**
 - Isso fará com que os manipuladores de eventos dos ancestrais não recebam o evento

```
<div>Um div com um <button>botão</button>.</div>

var bloco = document.querySelector("div");
var botao = document.querySelector("button");
bloco.addEventListener("mousedown", function() {
    console.log("Manipulador do bloco.");
});
botao.addEventListener("mousedown", function(event) {
    console.log("Manipulador do botão.");
    event.stopPropagation();
});
```

Ações Padrão

- Muitos eventos possuem ações padrão associadas a eles
 - Se você clica em um link, você será direcionado para o destino apontado pelo link
 - Se você preciona a tecla para baixo, o scroll do browser será ativado para baixo
 - Se você clica com o botão direito em uma página, o browser irá mostrar um menu de contexto
- Para a maioria dos eventos, os manipuladores de eventos são chamados antes da ação padrão

Ações Padrão

- É possível usar a função `preventDefault` para garantir que a ação padrão não seja executada

```
<a href="http://icomp.ufam.edu.br/">ICOMP</a>

var link = document.querySelector("a");
link.addEventListener("click", function(event) {
    console.log("Usuário não encaminhado.");
    event.preventDefault();
});
```

- Isto pode ser usado, por exemplo, para criar seus próprios atalhos de teclados ou menus de contexto

Eventos de Teclado

- O browser dispara um evento **keydown** quando uma tecla é clicada, e **keyup** quando a tecla é solta

```
<p>Esta página fica violeta quando V é pressionado</p>
```

```
addEventListener("keydown", function(event) {  
    if (event.keyCode == 86)  
        document.body.style.background = "violet";  
});  
  
addEventListener("keyup", function(event) {  
    if (event.keyCode == 86)  
        document.body.style.background = "";  
});
```

Eventos de Teclado

- Cliques em teclas tais como **Shift**, **Ctrl**, **Alt**, e **Meta** (Command do Mac) também geram eventos
- As propriedades **shiftKey**, **ctrlKey**, **altKey**, e **metaKey** são usadas para identificar tais cliques

```
<p>Pressione Ctrl-Space para continuar.</p>
```

```
addEventListener("keydown", function(event) {  
    if (event.keyCode == 32 && event.ctrlKey)  
        console.log("Continuando!");  
});
```

Eventos de Teclado

- Cliques em teclas tais como **Shift**, **Ctrl**, **Alt**, e **Meta** (Command do Mac) também geram eventos
- Além de **keydown** e **keyup**, também é possível usar o Evento **keypress**, que representa um caracter sendo digitado

```
addEventListener("keydown", function(event) {  
    if (event.keyCode == 32 && event.ctrlKey)  
        console.log("Continuando!");  
});
```

Eventos de Mouse

- **mousedown** e **mouseup** são similares aos eventos **keydown** e **keyup**, e são disparados no clique do mouse

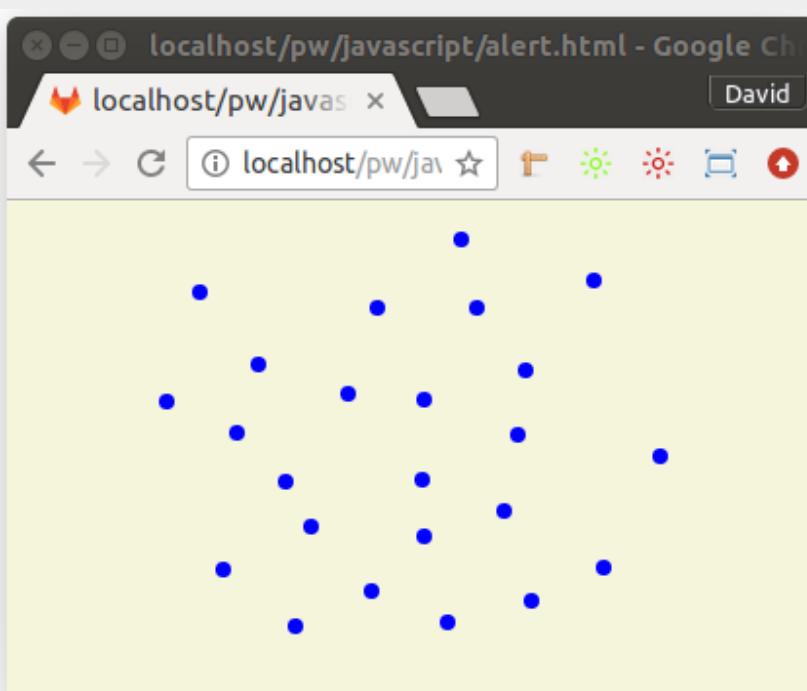
```
body {  
    height: 200px; background: beige;  
}  
.dot {  
    height: 8px; width: 8px; border-radius: 4px;  
    background: blue; position: absolute;  
}
```

```
addEventListener("mousedown", function(event) {  
    var dot = document.createElement("div");  
    dot.className = "dot";  
    dot.style.left = (event.pageX - 4) + "px";  
    dot.style.top = (event.pageY - 4) + "px";  
    document.body.appendChild(dot);  
});
```

Eventos de Mouse

- **mousedown** e **mouseup** são similares aos eventos **keydown** e **keyup**, e são disparados no clique do mouse

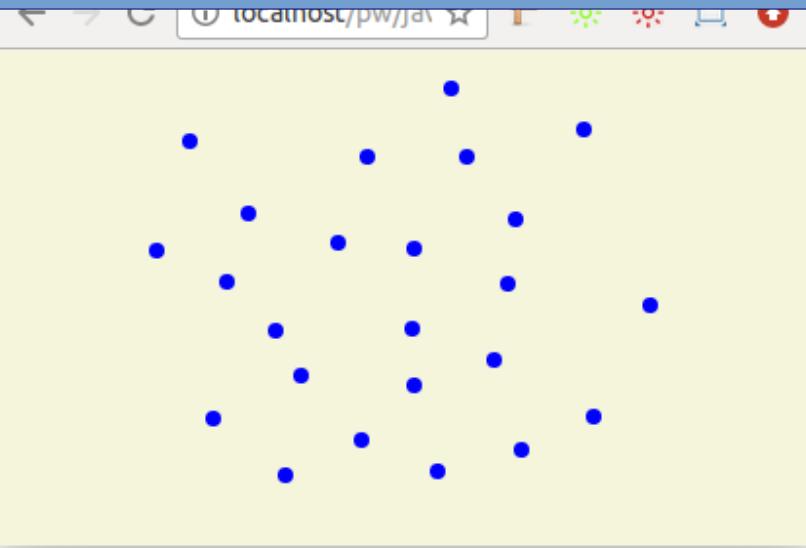
```
body {  
    height: 200px; background: beige;  
}  
.dot {  
    height: 8px;  
    background:  
}  
  
addEventListener('click', function(e) {  
    var dot = document.createElement('div');  
    dot.className = 'dot';  
    dot.style.left = e.clientX + 'px';  
    dot.style.top = e.clientY + 'px';  
    document.body.appendChild(dot);  
});
```



Eventos de Mouse

- **mousedown** e **mouseup** são similares aos eventos **keydown** e **keyup**, e são disparados no clique do mouse

Além de **mousedown** e **mouseup**, também existem os eventos **click** e **dblclick**



```
background: #f0f0ff;
}

addEventListener('click', function(e) {
    var dot = document.createElement('div');
    dot.className = 'dot';
    dot.style.left = e.clientX + 'px';
    dot.style.top = e.clientY + 'px';
    dot.style.width = '10px';
    dot.style.height = '10px';
    dot.style.backgroundColor = 'blue';
    document.body.appendChild(dot);
});
```

Outros Tipos de Eventos

- Movimentos do mouse
 - **Mousemove, mouseout, mouseover**
- Uso da barra de rolagem
 - **Scroll**
- Elementos em foco
 - **Focus, blur**
- Carregamento de página
 - **Load, beforeunload**

Trabalhando com Eventos

- Além da função **addEventListener**, existem outros métodos de se criar manipuladores de eventos
- Por exemplo, podemos usar atributos HTML de acordo com o tipo de evento desejado
 - O valor do atributo deve ser um conjunto de comandos JavaScript

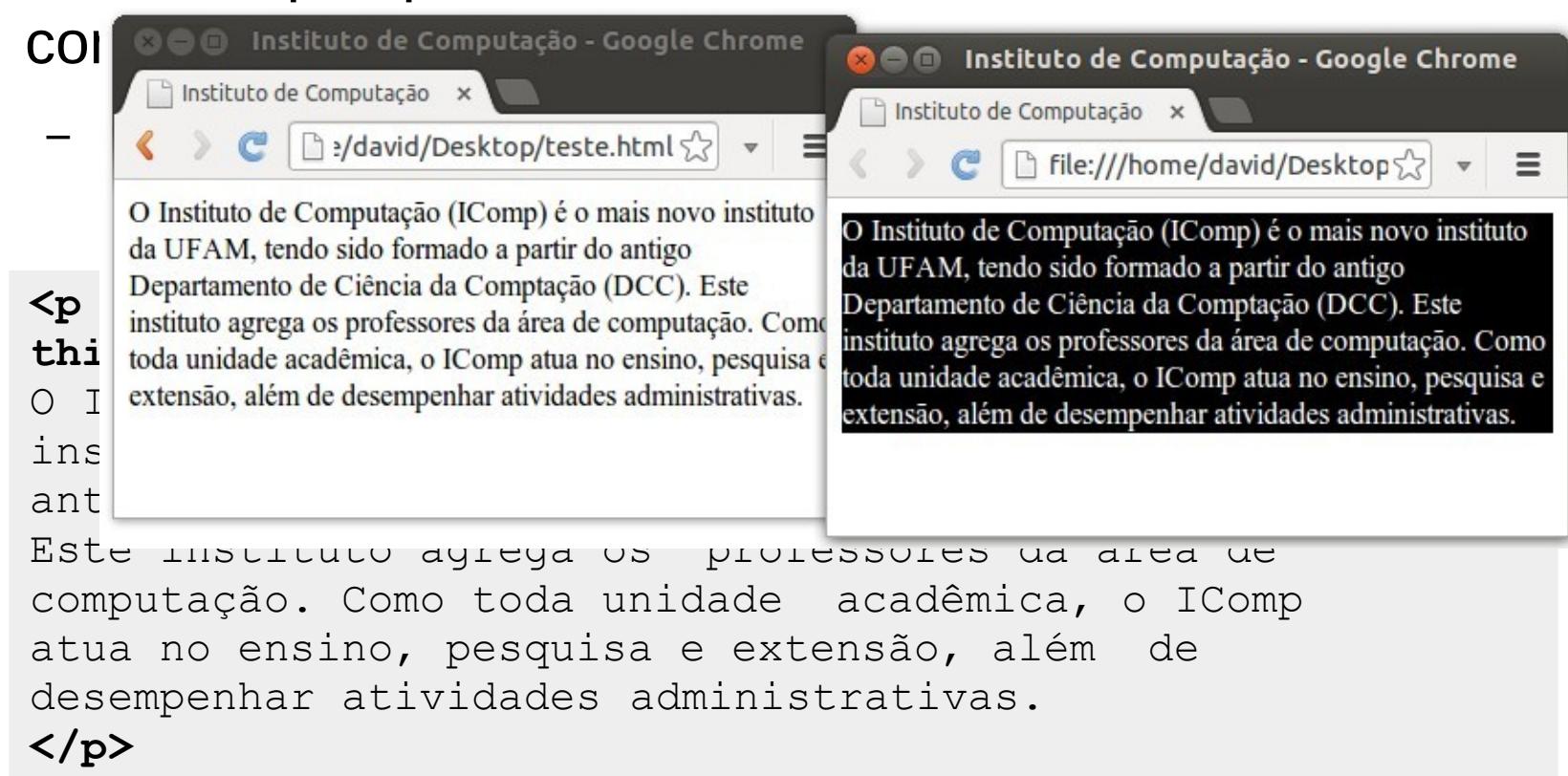
```
<p onmouseover="this.style.background='black';  
this.style.color='white'">
```

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

- Além da função **addEventListener**, existem outros métodos de se criar manipuladores de eventos
- Por exemplo, podemos usar atributos HTML de acordo



Trabalhando com Eventos

- Além da função **addEventListener**, existem outros métodos de se criar manipuladores de eventos
- Por exemplo, podemos usar atributos HTML de acordo com o tipo de evento desejado

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

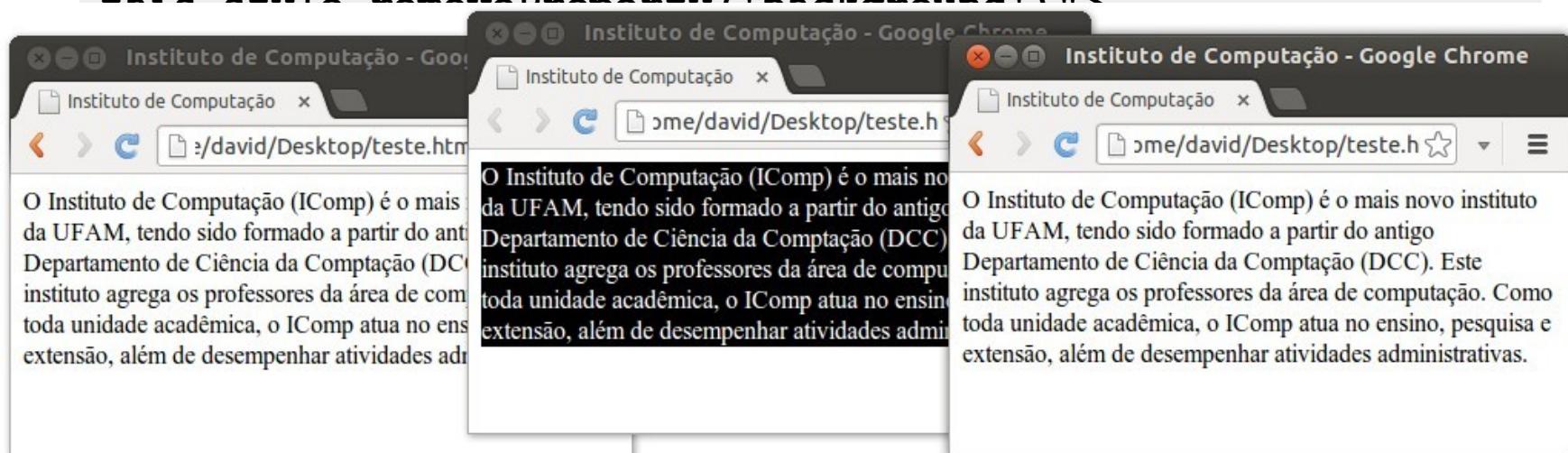
O Instituto de Computação (IComp) é o mais novo instituto da UFAM, tendo sido formado a partir do antigo Departamento de Ciência da Comptação (DCC). Este instituto agrupa os professores da área de computação. Como toda unidade acadêmica, o IComp atua no ensino, pesquisa e extensão, além de desempenhar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

```
</p>
```

Trabalhando com Eventos

- Além da função **addEventListener**, existem outros métodos de se criar manipuladores de eventos
- Por exemplo, podemos usar atributos HTML de acordo com o tipo de evento desejado

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



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 Comptação (DCC).  
Este instituto agrupa os professores da área de  
computação.  
</p>
```

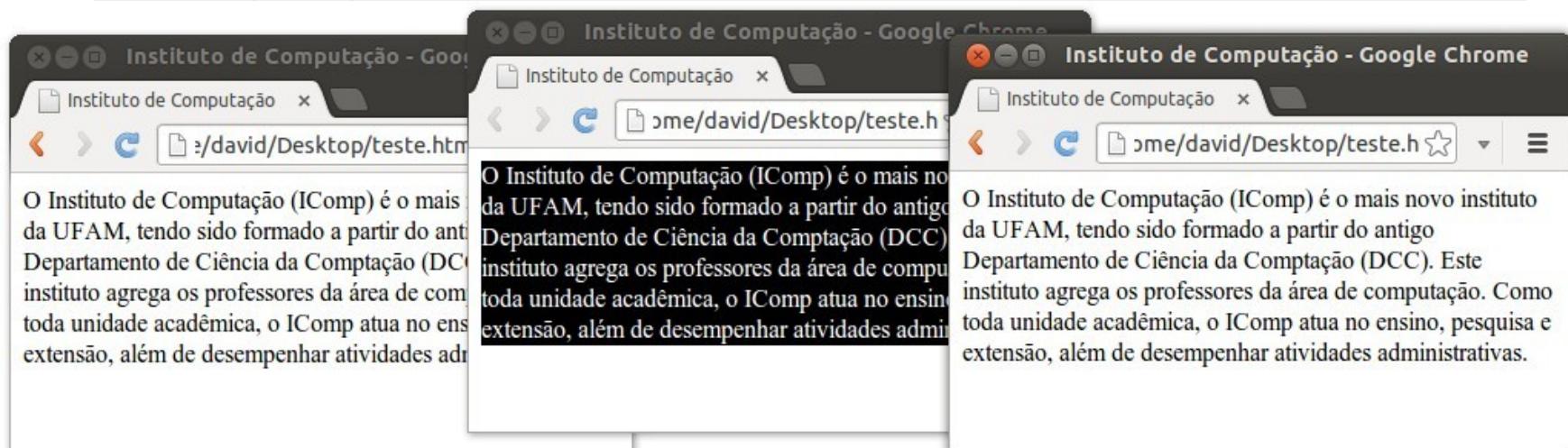
```
function handleMouseOver(elem) {  
    elem.style.background='black';  
    elem.style.color='white';  
}  
function handleMouseOut(elem {  
    elem.style.removeProperty('color');  
    elem.style.removeProperty('background');  
}
```

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 Comptação (DCC) .



```
elem.style.removeProperty('background');  
}
```

Trabalhando com Eventos

- Conforme dito anteriormente, também podemos usar propriedades dos elementos da DOM

```
<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';
}

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

Trabalho Prático de JS

- Desenvolver o jogo **snake** em javascript, similar ao disponível em <https://github.com/patorjk/JavaScript-Snake>
- Neste jogo, todos os movimentos da cobra deverão ser implementados através de um único **closure**

