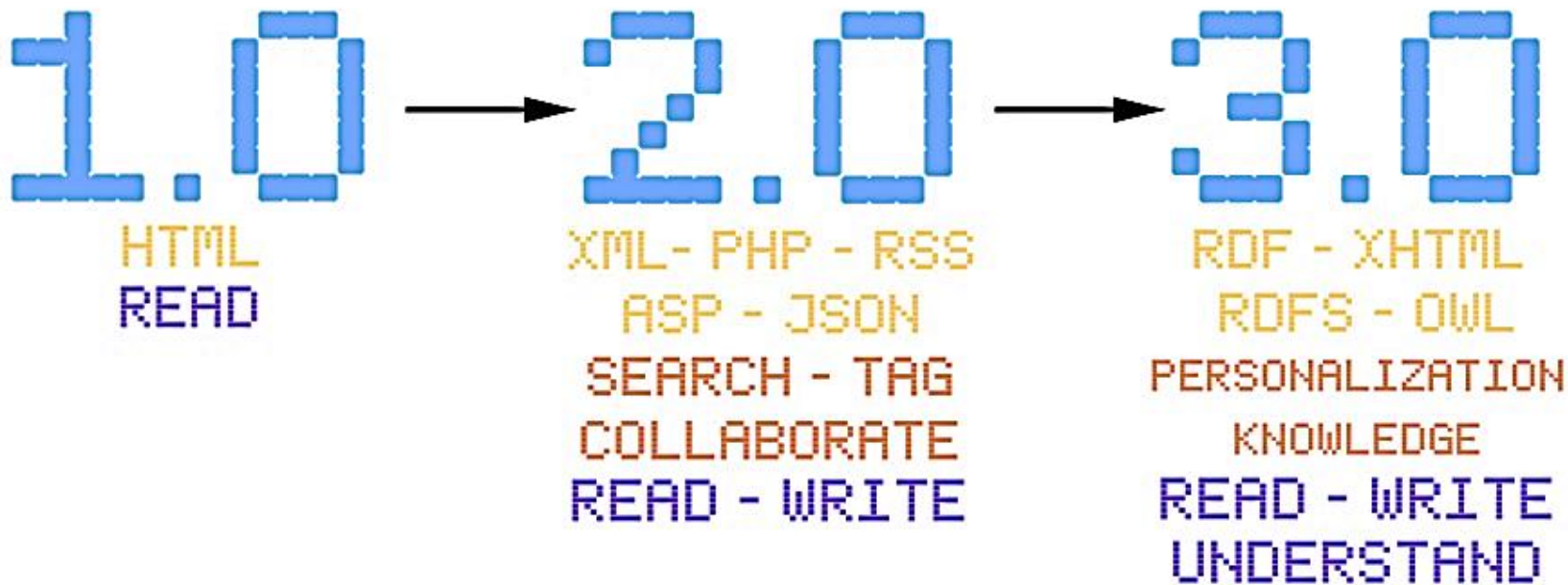


# **Evolução da Web**

Rommel Vieira Carneiro



# Evolução da Web



# O W3C – World Wide Web Consortium

O **World Wide Web Consortium (W3C)** é uma comunidade internacional que mantém e evolui os padrões da Web.

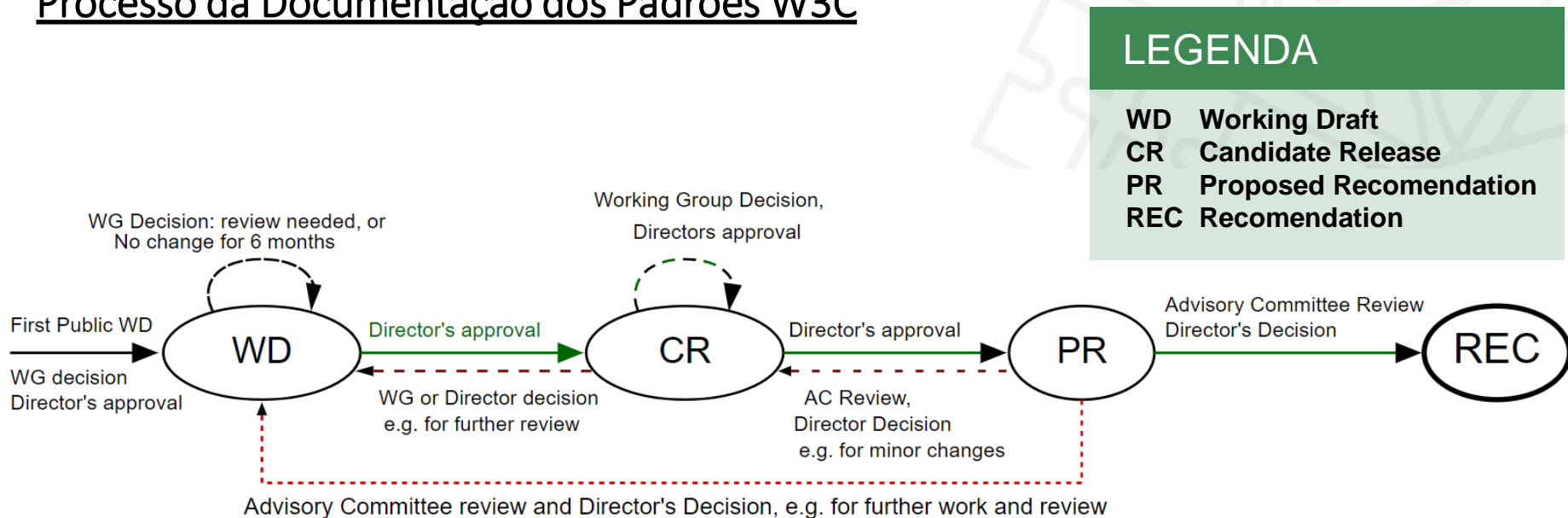
Padrões tais como:

- Design e Aplicações Web (HTML, CSS, SVG, Ajax, Acessibilidade);
- Arquitetura da Web (Protocolo HTTP, URI);
- Web Semântica (Linked Data - RDF, OWL, SPARQL);
- Tecnologia XML (XML, XML Schema, XSLT);
- Entre outros.



# O W3C – World Wide Web Consortium

## Processo da Documentação dos Padrões W3C



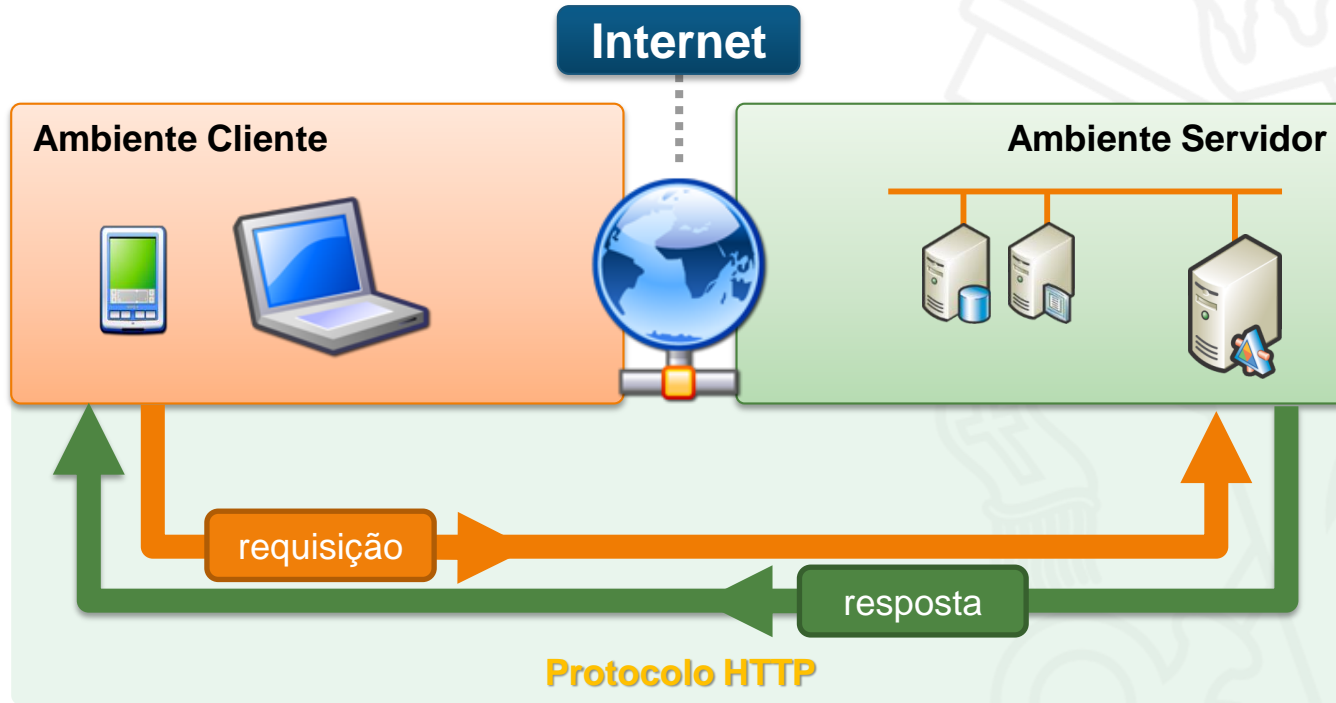


**PUC Minas**  
**Virtual**

# Arquitetura da Web

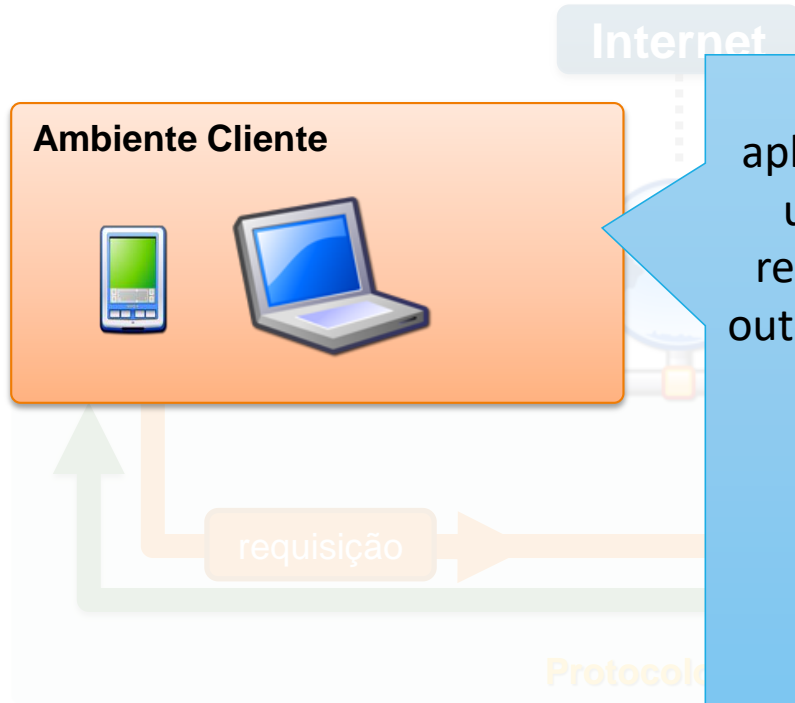
Rommel Vieira Carneiro

# Arquitetura da Web – Modelo Cliente-Servidor





# Arquitetura da Web – Modelo Cliente-Servidor



O **Cliente Web** é um programa ou aplicação específica, na maioria das vezes um Navegador ou Browser, que envia requisições via protocolo HTTP(S) a uma outra aplicação, o **Servidor Web** através de uma rede como a Internet.



# Arquitetura da Web – Modelo Cliente-Servidor

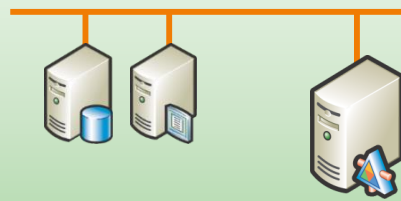
O **Servidor Web** é um programa que recebe requisições HTTP(S), interpreta a URL e em seguida envia resposta ao **Cliente Web** com o recurso solicitado (arquivo HTML, CSS, JavaScript, imagens, vídeos, folhas de estilo) por meio da rede.

NGINX



Internet

Ambiente Servidor



resposta

olo HTTP

# Arquitetura da Web – Modelo Cliente-Servidor



A **Internet** é uma rede mundial de computadores baseada no protocolo TCP/IP, onde todo computador conectado é denominado *host* (hospedeiro) e possui um identificador **endereço IP** (Internet Protocol) no padrão A.B.C.D (ex: 200.20.15.22).

Os seres humanos utilizam nomes como **www.pucminas.br** que são traduzidos em endereços IP antes que ocorra a comunicação.

# Arquitetura da Web – Modelo Cliente-Servidor

O **protocolo HTTP** é a forma como clientes e servidores se comunicam na rede. As **requisições** e as **respostas** obedecem aos padrões estabelecidos pelo protocolo HTTP.

A **requisição HTTP** é um pacote de dados enviado pela rede pelo **Cliente Web** para o **Servidor Web** e identifica o recurso solicitado. A resposta HTTP é formada por pacotes de dados enviados pelo **Servidor Web** para o **Cliente Web** com os recursos solicitados.





**PUC Minas**  
**Virtual**

# Protocolo HTTP

Rommel Vieira Carneiro

# Protocolo HTTP

- Histórico de Versões
- Requisição/Resposta
- Criptografia SSL/TLS

**http://**

# Protocolo HTTP – Histórico de Versões

- O HTTP 0.9 foi lançado em 1991
- O HTTP 1.0 foi lançado em 1996
- O HTTP 1.1 traz:
  - Conexões TCP persistentes diferentemente do HTTP 1.0
  - Suporte a Virtual Host (Cabeçalho Host)
  - Autenticação Digest
  - Possibilidade de compressão de dados
- Google propõe o SPDY em 2009
- O HTTP 2.0, lançado em 2015, trazendo:
  - Compressão de dados obrigatória
  - Cabeçalhos binários
  - Requisições paralelas
  - Envio apenas de cabeçalhos alterados nas próximas requisições
  - Priorização de requisições
  - Server PUSH – Envio automático de arquivos adicionais.



**http://**



# Protocolo HTTP – Requisição/Resposta

## Requisição

GET /index.html HTTP/1.1

Host: www.example.com

## Resposta

HTTP/1.1 200 OK

Date: Mon, 23 May 2005 22:38:34 GMT

Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)

Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT

Accept-Ranges: bytes

Content-Length: 438

Connection: close

Content-Type: text/html; charset=UTF-8

<html>

<head> <title>HTTP Response Example</title></head>

<body> Seja bem-vindo!! </body>

</html>

# Protocolo HTTP – Requisição/Resposta

Linha de  
Requisição

método

SPC

url

SPC

versão HTTP

CR

LF

Linhas de  
Cabeçalho

cabeçalho:

SPC

valor

CR

LF

■ ■ ■

cabeçalho:

SPC

valor

CR

LF

Linha em  
branco

CR

LF

Corpo da  
entidade

# Protocolo HTTP – Requisição/Resposta

**Linha de Resposta**

versão

SPC

código status

SPC

frase

CR

LF

**Linhas de Cabeçalho**

cabeçalho:

SPC

valor

CR

LF

■ ■ ■

cabeçalho:

SPC

valor

CR

LF

**Linha em branco**

CR

LF

**Corpo da entidade**

# Protocolo HTTP – Requisição/Resposta

## Requisição

```
GET /test/demo.asp?nome=Fulano&curso=Computacao HTTP/1.1  
Host: www.site.com.br
```

## Resposta

```
HTTP/1.1 200 OK  
Server: Microsoft-IIS/4.0  
Date: Mon, 3 Jan 1998 13:13:33 GMT  
Content-Type: text/html  
Last-Modified: Mon, 11 Jan 1998 13:23:42 GMT  
Content-Length: 112
```

```
<html>  
<head><title>HTTP Response Example</title></head>  
<body> Fulano, seja bem-vindo a Computacao</body>  
</html>
```

# Protocolo HTTP – Requisição/Resposta

GET: requisita dados de um recurso específico

```
GET /test/go.asp?nome=Fulano&curso=Computacao HTTP/1.1  
Host: www.site.com.br
```

POST: Envia informações para o servidor

```
POST /test/go.asp HTTP/1.1  
Host: www.site.com.br  
Accept: text/html  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 30  
nome=Fulano&curso=Computacao
```

HEAD: Recupera apenas cabeçalho HTTP das respostas do servidor

```
HEAD /test/go.asp?nome=Fulano&curso=Computacao HTTP/1.1  
Host: www.site.com.br
```

# Protocolo HTTP – Requisição/Resposta

## Códigos de retorno

Código	Propósito	Descrição
1xx	Informacional	Requisição recebida, processo em continuidade
2xx	Sucesso	A ação foi recebida, entendida e aceita
3xx	Redirecionamento	Ações adicionais devem ser executadas para completar o pedido
4xx	Erro no cliente	O pedido contém erro de sintaxe ou não pode ser completado
5xx	Erro no servidor	O servidor falhou em completar um pedido aparentemente válido

## Exemplos mais comuns

- 200 – Ok
- 403 – Acesso negado
- 404 – Página não encontrada
- 500 – Erro interno do servidor



# Protocolo HTTP – Criptografia SSL/TLS

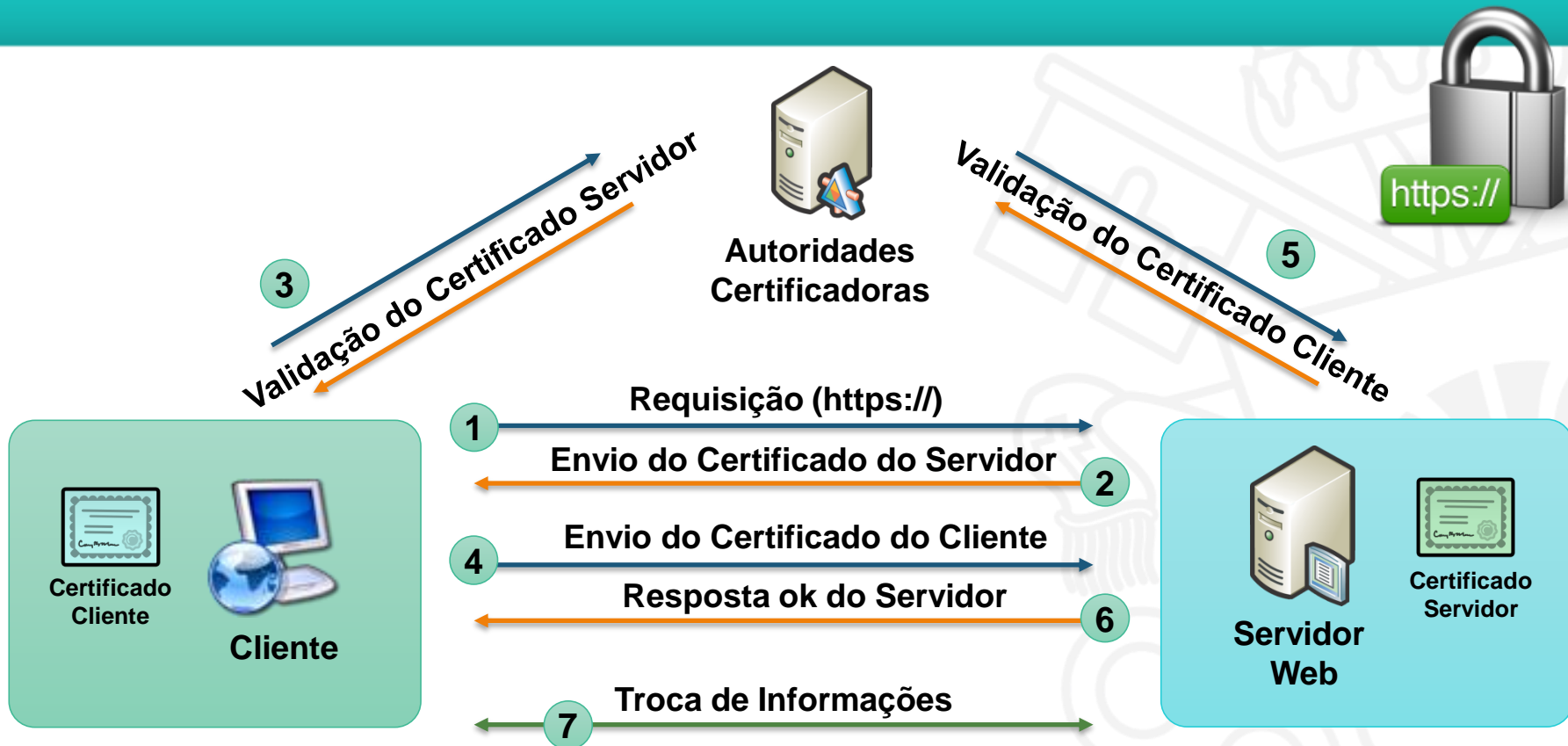
HTTPS identifica a comunicação segura por meio do protocolo HTTP, na porta 443 (por padrão), utilizando os protocolos TLS ou SSL.



## Características

- Fornece conexão criptografada com identificação de cliente e servidor
- Baseado em certificados digitais emitidos por autoridades certificadoras
- Requer que servidores Web sejam configurados com certificados digitais
- Requer que os navegadores reconheçam as autoridades certificadores emissoras dos certificados do servidor

# Protocolo HTTP – Criptografia SSL/TLS





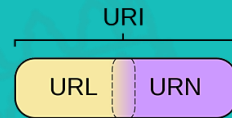


**PUC Minas**  
**Virtual**

# URI, URN e URL

Rommel Vieira Carneiro

# URI, URL e URN

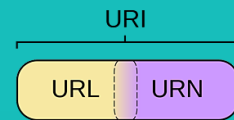


**URI (Uniform Resource Identifier)** é um padrão para endereçamento que identifica de forma precisa recursos disponíveis na rede.

A URI engloba os conceitos de:

- **URL** (Uniform Resource Locator)
- **URN** (Uniform Resource Name)

# URI, URL e URN



`ftp://exemplo.com:8042/over/there?name=ferret#nose` **URL**

esquema

autoridade

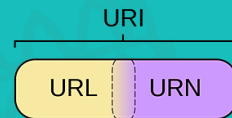
caminho

query

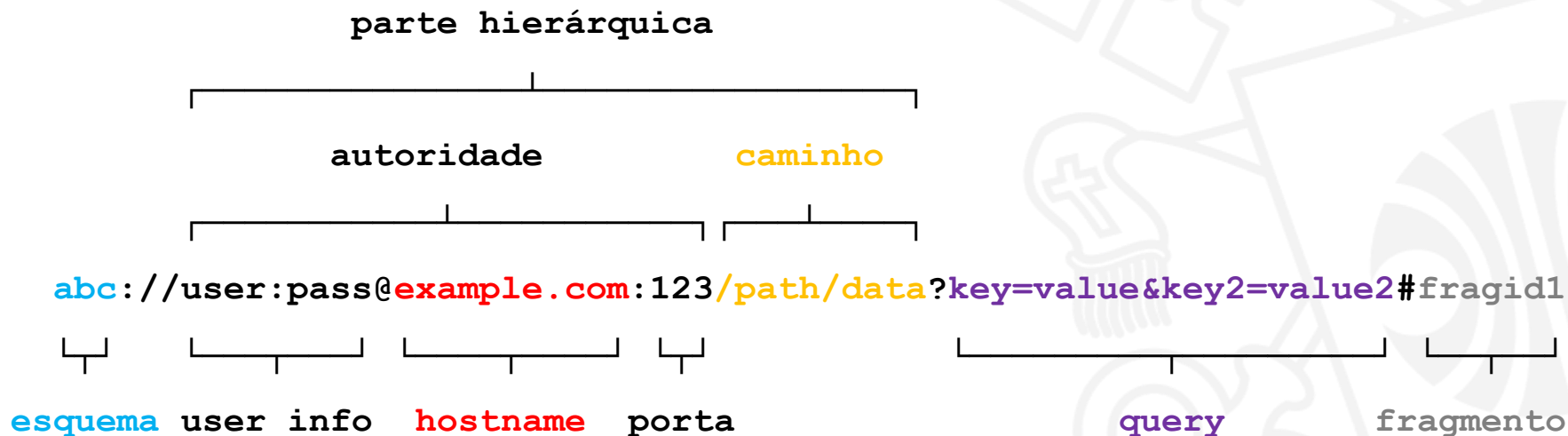
fragmento

`urn:example:animal:ferret:nose` **URN**

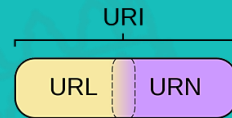
# URI, URL e URN



URL (Uniform Resource Locator) é um padrão de URI que serve para referenciar um recurso e sua localização, normalmente na Internet.



# URI, URL e URN

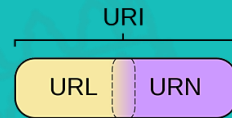


## Estrutura de um URL

- **Esquema** – forma de interação entre cliente e servidor, Ex: http, ftp, entre outros
- **User:pass** – informações de usuário
- **Hostname** – nome ou número IP onde se encontra a aplicação servidor
- **Porta** – porta TCP no servidor. Por padrão, HTTP usa a porta é 80 e pode ser omitida
- **Caminho** – indica o local exato onde o recurso se encontra
- **Query** – dados não hierárquicos de uma consulta sob a forma de pares nome e valor
- **Fragmento** – identifica uma seção no recurso

**esquema://user:pass@hostname:porta/caminho?query#fragmento**

# URI, URL e URN



**URN (Uniform Resource Name)** é um padrão de URI que identifica um recurso (NSS) pelo nome em um determinado *namespace* (NID).

**urn:example:mammal:monotreme:echidna**

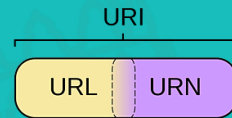
└──┬──────────────────────────────────┘

esquema namespace (NID + NSS)

Legenda:

- NID – Namespace Identifier
- NSS – Namespace Specific String

# URI, URL e URN



## Exemplos de URI

URL → `esquema://user:pass@host:porta/caminho?query#fragment`

- `http://localhost:80/admin/index.php?z1-w1&z2=w2`
- `ftp://user:pass@server.net:21/documentos/arquivo.zip`
- `news://pl.com.os.linux`
- `telnet://192.168.1.1`

URN → `esquema:namespace_identificier:namespace_specific_string`

- `urn:isbn:978-1-491-91866-1`





**PUC Minas**  
**Virtual**

# Linguagem HTML

Rommel Vieira Carneiro

# Linguagem HTML



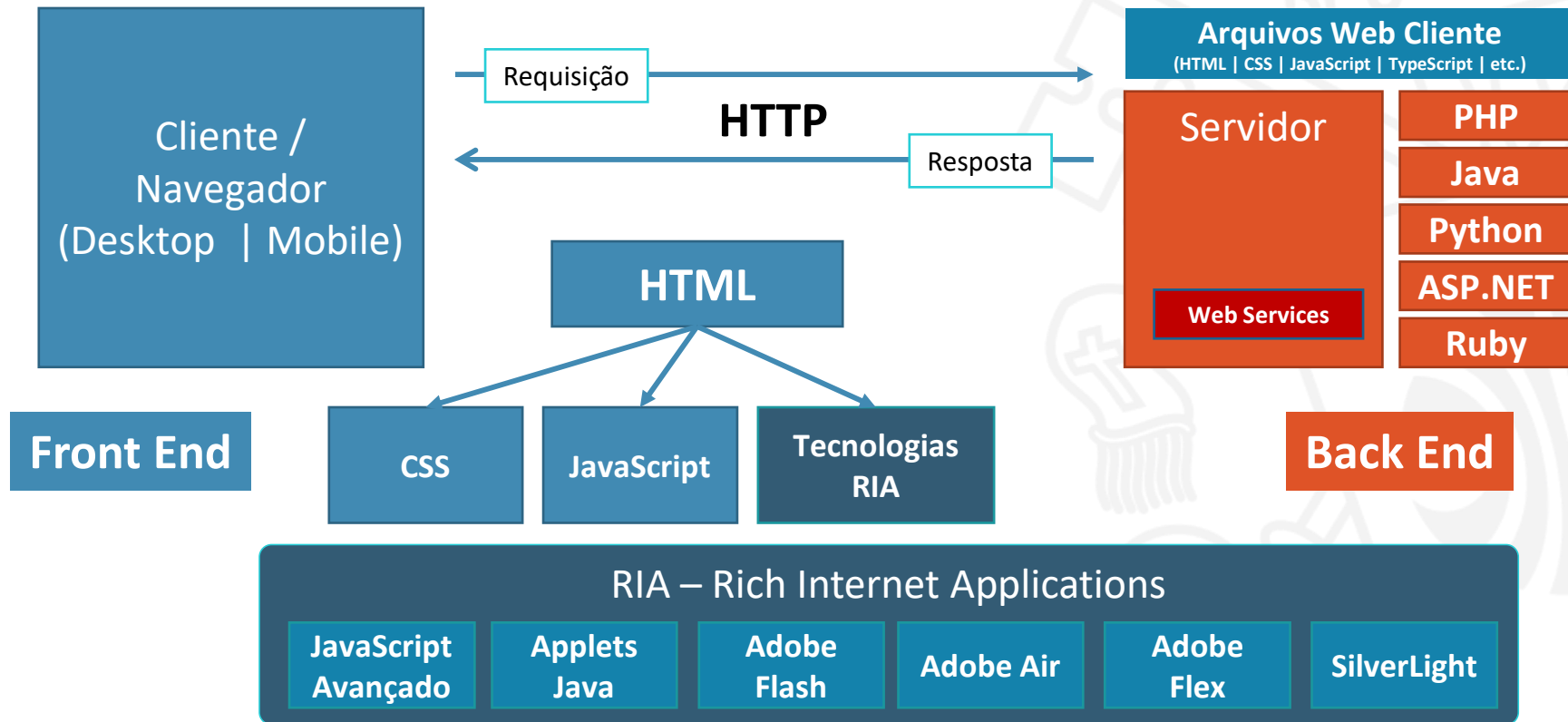
## O que é HTML?

HTML ou *HyperText Markup Language* é a **linguagem de marcação**, baseada em tags (ou marcas), utilizada para produção de documentos ou páginas Web, no **formato de hipertextos**, que são interpretados pelos Navegadores.

Os **hipertextos** são documentos que utilizam hiperlinks para outros documentos relacionados, daí o nome Web (teia).

A linguagem HTML foi inicialmente uma aplicação do **padrão SGML (*Standard Generalized Markup Language*)**, utilizado para definir linguagens de marcação. Na versão 5, isto foi abandonado.

# Ambiente de Aplicações Web



# HTML – Estrutura do Documento

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="author" content="Rommel">
  </head>

  <body>
    <h1> titulo </h1>
    <p> paragrafo </p>
  </body>
</html>
```

# HTML – Estrutura do Documento

atributos →  
**nome**="valor"

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta name="author" content="Rommel">  
  </head>  
  <body>  
    <h1> titulo </h1>  
    <p> paragrafo </p>  
  </body>  
</html>
```

versão html

cabeçalho

corpo

documento

tags → <tag> [conteúdo] </tag>

# HTML – Sintaxe – Elementos HTML

## Elementos HTML

Os elementos são a estrutura básica do documento HTML, marcados por meio de tags que são delimitadas pelos símbolos < e >. Veja o exemplo:

```
<body> [conteúdo] </body>
```

Tag de abertura </body>

Tag de fechamento </body>

Toda página deve estar contida dentro de um elemento html.

```
<html> [conteúdo] </html>
```

Os nomes dos elementos nas tags **NÃO** são sensíveis ao case (maiúsculo ou minúsculo é indiferente).

# HTML – Sintaxe – Elementos HTML

Elementos HTML - Podem se apresentar em quatro formatos:

- Elementos com elementos filhos

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

- Elementos com texto

```
<title> PUC Minas Web Site </title>
```

- Elementos vazios:

```
<meta name="author" content="Rommel"> ou <br>
```

- Elementos de conteúdo misto (texto e elementos filhos)

```
<p> Documento <span lang="en">Web</span> </p>
```



# HTML – Sintaxe – Atributos

## Atributos

- Os atributos podem ser incluídos em elementos HTML.
- Um atributo não se repete em um elemento.
- São definidos pelo par nome/valor, podendo ter valor nulo

```
<input disabled name="usuario" value="rommelcarneiro">
```

- Recomenda-se utilizar aspas duplas.
- Os atributos alteram o funcionamento dos elementos do HTML.
- Cada elemento possui um conjunto próprio de atributos.
- Os atributos possuem valores livres ou pré-definidos.

# HTML – Estrutura do Documento – Comentários

## Comentários

- Os comentários não são exibidos pelo navegador
- São delimitados por `<!--` e `-->`

**`<!-- Os comentários documentam o código HTML -->`**

# HTML – Estrutura do Documento – DOCTYPE

O **DOCTYPE** indica ao browser qual o versão do HTML está sendo utilizada no documento, alternando entre *quirks mode* e *strict mode*.

## HTML5

```
<!DOCTYPE html>
```

```
<!DOCTYPE html>
```

## HTML 4.01

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">
```


## XHTML 1.0

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

# HTML – Estrutura do Documento – Cabeçalho

**Cabeçalho** - Primeira parte do arquivo HTML, representada pela tag **<head>**, que inclui informações sobre o documento (metadados), referências a scripts, folhas de estilo (CSS) que complementam o documento.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <title>Página de Exemplo</title>
  <meta name="description" content="PUC Minas Web site">
  <meta name="author" content="Rommel">
  <meta name="keywords" content="html, web, css">
  <link rel="stylesheet" href="style.css">
</head>
<body> ... </body>
</html>
```



# HTML – Estrutura do Documento – Cabeçalho

## Cabeçalho – Tags mais comuns

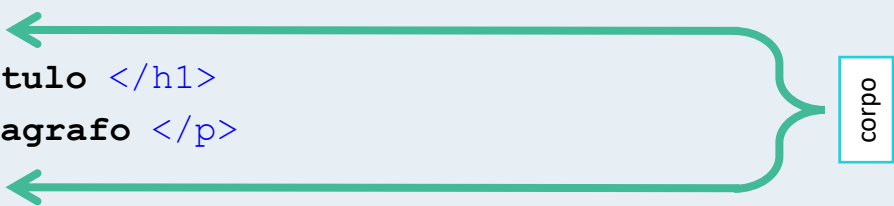
- **title**: Define o título do documento
- **style**: Define códigos de formatação no padrão CSS
- **script**: Define códigos de scripts em alguma linguagem própria
- **link**: Definem ligações com outros arquivos como: CSS, scripts, etc.
- **meta**: informações sobre o documento tais como codificação de caracteres, descrição, palavras-chave, autor, etc.

```
<head>
  <title>Página de Exemplo</title>
  <meta name="description" content="PUC Minas Web site">
  <meta name="author" content="Rommel">
  <meta name="keywords" content="html, web, css">
  <link rel="stylesheet" href="style.css">
</head>
```

# HTML – Estrutura do Documento – Corpo

**Corpo** - Segunda parte do arquivo HTML, representada pela tag **<body>**, que inclui todo o conteúdo da página exibido ao usuário pelo Navegador. É composto por textos, links, imagens, vídeos, tabelas, formulários.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head> ... </head>
  <body>
    <h1> Título </h1>
    <p> paragrafo </p>
  </body>
</html>
```





**PUC Minas**  
**Virtual**

# Linguagem CSS

Rommel Vieira Carneiro



# Linguagem CSS

CSS



O Cascading Style Sheet (CSS) é uma linguagem para definição de regras de apresentação em documentos HTML e XML.

Seu objetivo é promover a separação entre o formato e o conteúdo em um documento.

## Page.html

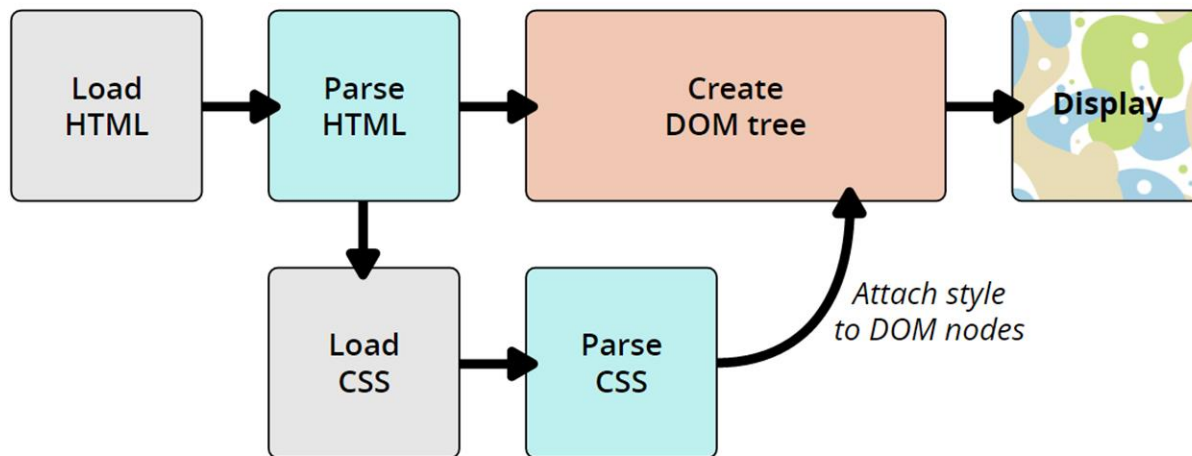
```
<html>
  <head>
    <link rel="stylesheet"
          href="style.css">
  </head>
  <body>
    <h1>Page</h1>
    <p> Hello World! </p>
  </body>
</html>
```

## Style.css

```
body{
  background-color:#eeeeee;
  color: #111100
  font-family: Arial,
  Tahoma;
  font-size: 10px;
}
h1 {
  color: red;
}
```

# CSS – Como funciona?

## Dinâmica de carga e processamento



# CSS – Como funciona?

## Page.html

```
<html>
  <head>
    <link rel="stylesheet"
          href="style.css">
  </head>
  <body>
    <h1>Page</h1>
    <p>Hello World!</p>
  </body>
</html>
```



## Style.css

```
body{
  background-color:yellow;
  color: blue;
  font-family: Arial;
  font-size: 20px;
}
h1 {
  color: red;
}
```

**Page**

Hello World!

**Page**

Hello World!

# CSS – Formas de utilização em páginas HTML

```
<html>
  <head>
    <link rel="stylesheet" href="style.css" type="text/css">
    <title>Exemplo CSS</title>
    <style type="text/css">
      p{ font-size: 10pt; font-family: "Verdana"; color: blue; }
      h1{ font-size: 16pt; font-family: "Impact"; color: red }
    </style>
  </head>
  <body>
    <p style="margin-left: 0.5in; font-size: 8pt">...</p>
    <span style="font-weight: bold; background: red">...</span>
  </body>
</html>
```

Arquivo CSS  
Externo

Bloco  
interno

Inline

Prioridade de Aplicação

1) Inline

2) Bloco  
interno

3) Arquivo CSS  
externo

4) Browser  
Default

# CSS – Formas de utilização em páginas HTML

## Arquivo CSS Externo

- Considerada a forma mais recomendada de implementação
- Separa completamente formato e conteúdo
- Permite atualizar as definições de CSS em um único lugar

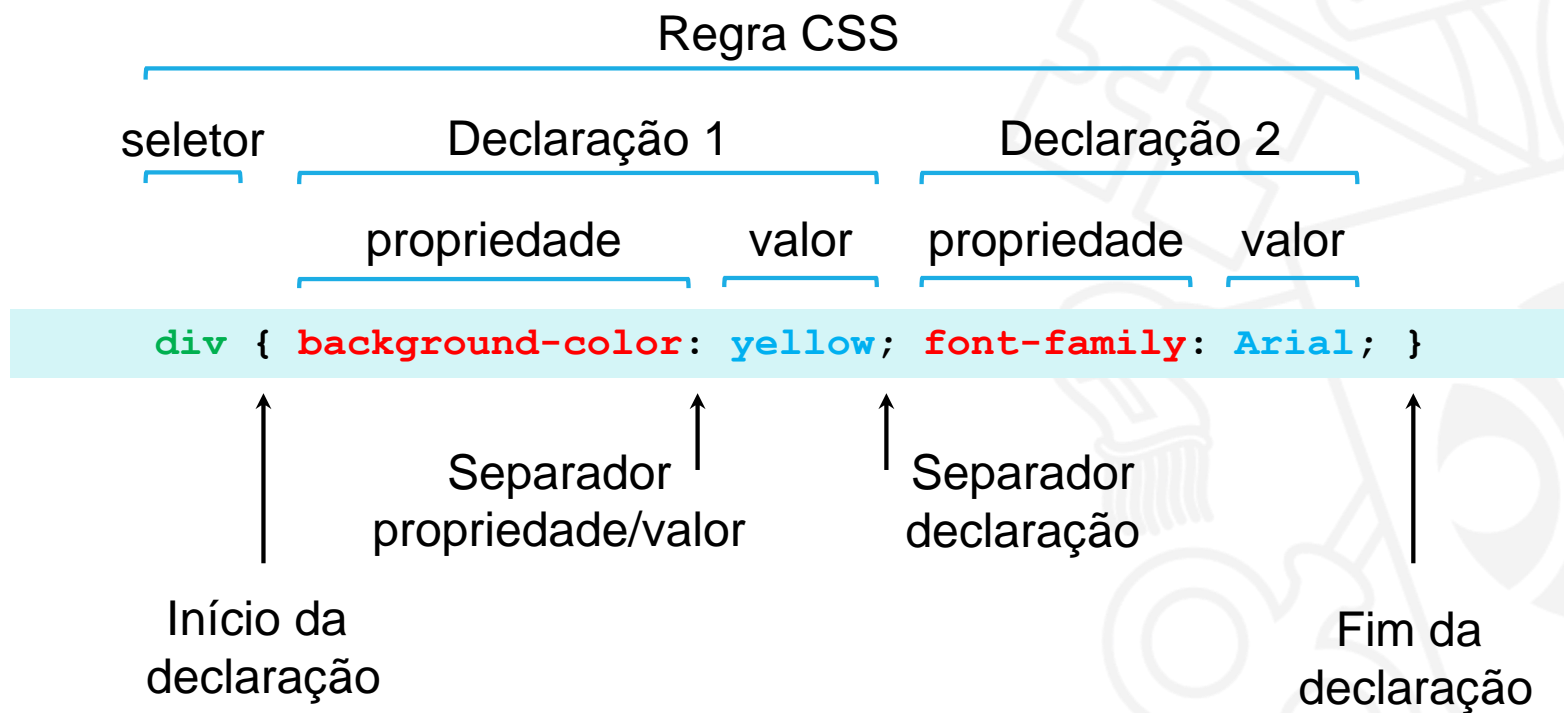
## Bloco interno

- Útil em algumas circunstâncias, com regras locais para um conteúdo em específico
- Requer a atualização em todos os locais em que for usado

## Inline

- Não utilize este formato, a menos que não tenha alternativa
- Requer a atualização em diversos locais no mesmo arquivo
- Dificulta a legibilidade do HTML e mistura conteúdo e formatação

# CSS – Sintaxe das Regras



# CSS – Tipos de Seletores

## Tipos de seletores – Elemento

O seletor indica o elemento HTML nos quais serão aplicadas as declarações.

```
<p>Texto básico</p>  
<p id="p101">Texto com id</p>  
<p class="marked">Texto com class</p>  
<p lang="pt-br">Texto com atributo</p>
```

**Page.html**

```
p { color: blue; }
```

**Style.css**

### Tipos de Seletores

#### Elemento

Identificador (id)  
Classe (class)  
Atributo  
Pseudo-Classe  
Pseudo-Elemento  
Universal

### Resultado

Texto básico  
Texto com id  
Texto com class  
Texto com atributo

# CSS – Tipos de Seletores

## Tipos de seletores – Identificador (id)

O seletor indica o elemento HTML nos quais serão aplicadas as declarações.

```
<p>Texto básico</p>  
<p id="p101">Texto com id</p>  
<p class="marked">Texto com class</p>  
<p lang="pt-br">Texto com atributo</p>
```

**Page.html**

```
#p101 { color: blue; }
```

**Style.css**

### Tipos de Seletores

Elemento

**Identificador (id)**

Classe (class)

Atributo

Pseudo-Classe

Pseudo-Elemento

Universal

### Resultado

Texto básico

Texto com id

Texto com class

Texto com atributo



# CSS – Tipos de Seletores

## Tipos de seletores – Classe (class)

O seletor indica os elementos HTML que são de determinada classe.

```
<p>Texto básico</p>
<p id="p101">Texto com id</p>
<p class="marked">Texto com class</p>
<p lang="pt-br">Texto com atributo</p>
```

Page.html

```
.marked { color: blue; }
```

Style.css

### Tipos de Seletores

Elemento

Identificador (id)

**Classe (class)**

Atributo

Pseudo-Classe

Pseudo-Elemento

Universal

### Resultado

Texto básico

Texto com id

Texto com class

Texto com atributo

# CSS – Tipos de Seletores

## Tipos de seletores – Atributo

O seletor indica os elementos HTML que atendem a uma condição nos atributos.

```
<p>Texto básico</p>  
<p id="p101">Texto com id</p>  
<p class="marked">Texto com class</p>  
<p lang="pt-br">Texto com atributo</p>
```

**Page.html**

```
[lang] { color: blue; }  
[id="p101"] { color: green; }  
[class~="marked"] { color: red; }
```

**Style.css**

### Tipos de Seletores

Elemento

Identificador (id)

Classe (class)

**Atributo**

Pseudo-Classe

Pseudo-Elemento

Universal

### Resultado

Texto básico

Texto com id

Texto com class

Texto com atributo

# CSS – Tipos de Seletores

## Tipos de seletores – Pseudo-Classe

O seletor indica estados ou situações de elementos HTML.

```
<p>Texto básico</p>
<p id="p101">Texto com id</p>
<p class="marked">Texto com class</p>
<p lang="pt-br">Texto com atributo</p>
```

Page.html

```
p:first-of-type { color: orange; }
p:nth-child(3) { color: red; }
p:hover { color: green; }
```

Style.css

### Tipos de Seletores

Elemento

Identificador (id)

Classe (class)

Atributo

**Pseudo-Classe**

Pseudo-Elemento

Universal

### Resultado

Texto básico

Texto com id

Texto com class

Texto com atributo



# CSS – Tipos de Seletores

## Tipos de seletores – Pseudo-Classe

Indica estados ou situações de elementos HTML.

Pseudo-Class	Descrição
:first-of-type	Primeiro elemento de determinado tipo
:last-of-type	Último elemento de determinado tipo
:hover	Acontece quando o mouse passa sobre
:focus	Acontece quando elemento está em foco
:visited	Acontece com links já visitados
:link	Condição original dos links
:active	Acontece quando o elemento é ativado
:nth-child(n)	Enésimo elemento (n)
:first-child	Primeiro elemento filho
:last-child	Último elemento filho
:empty	Elemento vazio ou sem texto
:not()	Negação de um elemento. Ex. :not(p)

### Tipos de Seletores

Elemento

Identificador (id)

Classe (class)

Atributo

**Pseudo-Classe**

Pseudo-Elemento

Universal

Consulte outras pseudo-classes

# CSS – Tipos de Seletores

## Tipos de seletores – Pseudo-Elemento

O seletor indica partes de elementos HTML que podem ser manipuladas.

### Page.html

```
<p>Texto básico<br>próxima linha</p>
```

### Style.css

```
p::first-letter{ font-size:xx-large; }  
P::first-line { color: red; }  
P::after { content: "\00a9" }
```

### Tipos de Seletores

Elemento

Identificador (id)

Classe (class)

Atributo

Pseudo-Classe

**Pseudo-Elemento**

Universal

### Resultado

**T**exto básico  
próxima linha©

# CSS – Tipos de Seletores

## Tipos de seletores – Universal

Indica partes de elementos HTML que podem ser manipuladas.

```
<p>Texto básico</p>  
<p id="p101">Texto com id</p>  
<p class="marked">Texto com class</p>  
<p lang="pt-br">Texto com atributo</p>
```

**Page.html**

```
* { color: red }
```

**Style.css**

### Tipos de Seletores

Elemento

Identificador (id)

Classe (class)

Atributo

Pseudo-Classe

Pseudo-Elemento

**Universal**

### Resultado

Texto básico  
Texto com id  
Texto com class  
Texto com atributo

# CSS – Combinação de seletores

Regra CSS	Significado
<b>A, B { ... }</b>	Equivale a duas regras distintas A { } e B { } Ex: <code>p, span { color: black; }</code>
<b>AB { ... }</b>	Elementos associados a A e B ao mesmo tempo. Uso com classes, IDs. Ex: <code>p.titulo { color: black; }</code>
<b>A B { ... }</b>	Elementos associados a B, <b>descendentes</b> de elementos associados a A. Ex: <code>div p { color: black; }</code>
<b>A &gt; B { ... }</b>	Elementos associados a B, <b>filhos</b> de elementos associados a A Ex: <code>div &gt; p { color: black; }</code>
<b>A + B { ... }</b>	Elementos associado a B, <b>próximo irmão</b> de elementos associado a A Ex: <code>div + p { color: black; }</code>
<b>A ~ B { ... }</b>	Elementos associados a B, <b>próximos irmãos</b> de elementos associados a A Ex: <code>div ~ p { color: black; }</code>

# CSS – Comentários

Utilizados para documentação do código CSS e são ignorados pelo browser durante o processamento.

```
/* Regras do Box Model */  
div {  
    height: 100px;  
    background-color: red;  
    margin: 35px;  
    padding: 35px;  
    /* border: solid 20px black; */  
}
```

Documentação do código

Cancelar uma regra ou  
declaração específica



# CSS – Prioridade de Seletores

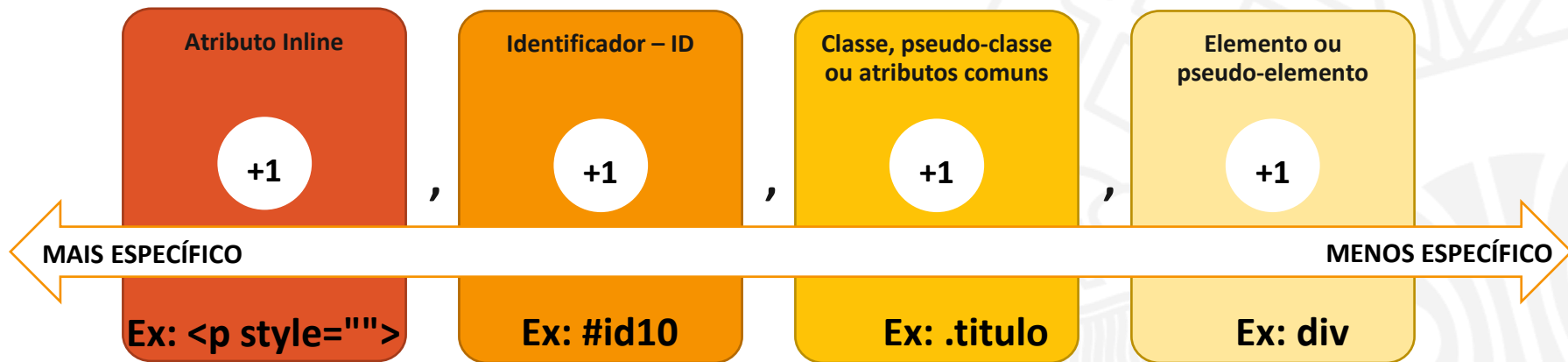
A escolha da declaração CSS a ser aplicada, no caso de conflitos, obedece as seguintes regras:

- Especificidade
- Ordem de leitura
- Importância



# CSS – Prioridade de Seletores – Especificidade

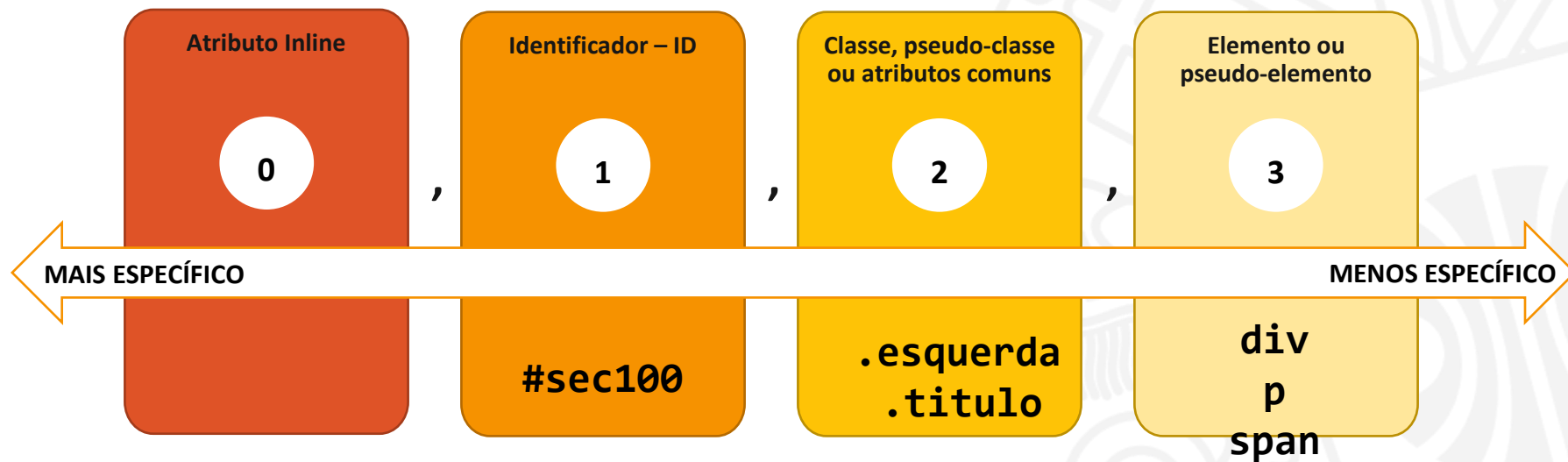
No cálculo da especificidade de uma regra, contabiliza-se as ocorrências de 4 itens:



Para cada item encontrado (inline style, ids, classes, elementos), contabiliza-se um ponto no respectivo aspecto.

# CSS – Prioridade de Seletores – Especificidade

**#sec100**   **div.esquerda**   **p**   **span.titulo**   { color:red }



**Especificidade resultante: 0,1,2,3**

# CSS – Prioridade de Seletores – Especificidade

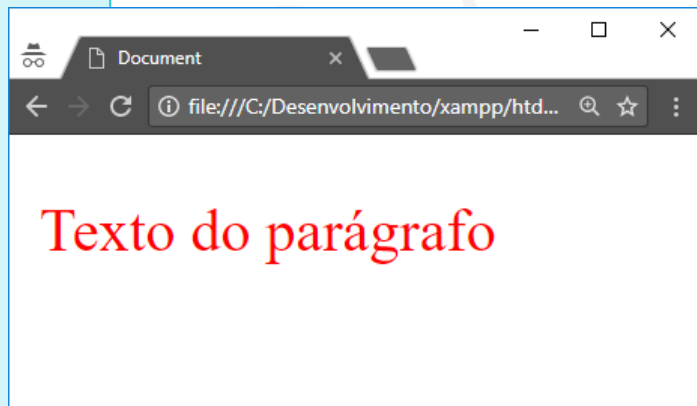
## Prioridade de Seletores – Cálculo de Especificidade – Exemplos

Seletor	Pontuação	Descrição
div ul li a	0,0,0,4	4 elementos, 4 pontos na primeira casa
div.content ul li	0,0,1,3	Uma classe vale um ponto na segunda casa. Mais 3 elementos, mais 3 pontos na primeira casa.
a:hover	0,0,1,1	Um elemento, um ponto na primeira casa. Mais uma pseudo-classe, que equivale a uma classe e logo ganha um ponto na segunda casa.
div.menu a:hover	0,0,2,2	Dois elementos, dois pontos na primeira casa. Mais uma classe e uma pseudo-classe, mais dois pontos na segunda casa.
#content p	0,1,0,1	Um ID equivale a um ponto na terceira casa. Mais um elemento, que equivale a um ponto na primeira casa.

# CSS – Prioridade de Seletores – Ordem de Leitura

Para regras concorrentes de mesma especificidade a última irá prevalecer.

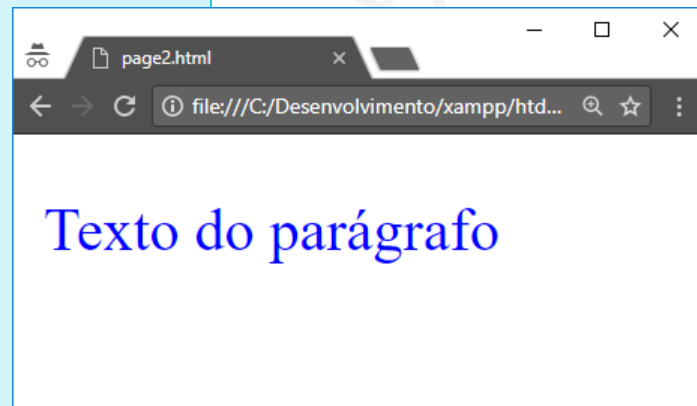
```
<!DOCTYPE html>
<html>
<head>
  <style>
    p { color: blue }
    p { color: green }
    p { color: red }
  </style>
</head>
<body>
  <p>Texto do parágrafo</p>
</body>
</html>
```



# CSS – Prioridade de Seletores – Importância

A palavra-chave **!important** faz a declaração prevalecer sobre as demais

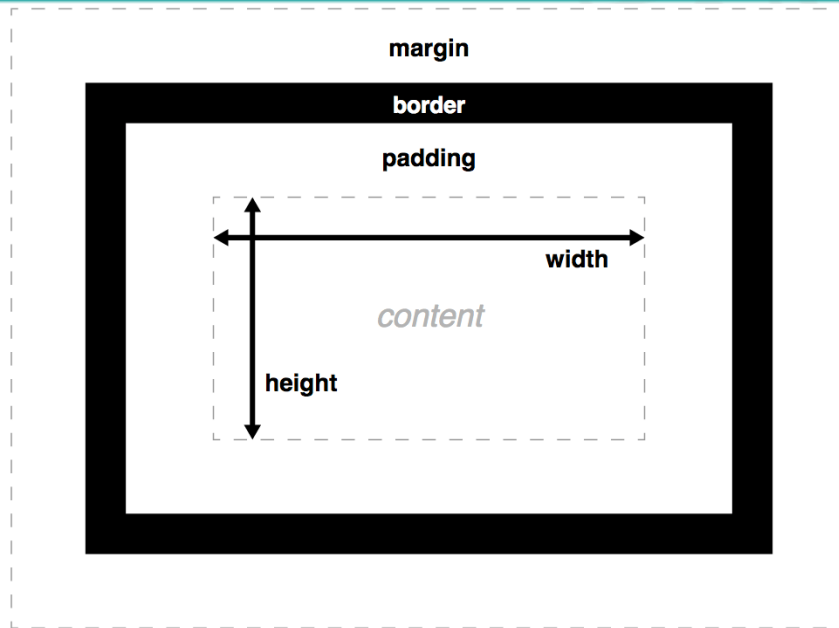
```
<!DOCTYPE html>
<html>
<head>
  <style>
    p { color: blue !important }
    p { color: green }
    p { color: red }
  </style>
</head>
<body>
  <p>Texto do parágrafo</p>
</body>
</html>
```



# CSS – Box Model

## Propriedades do Box Model

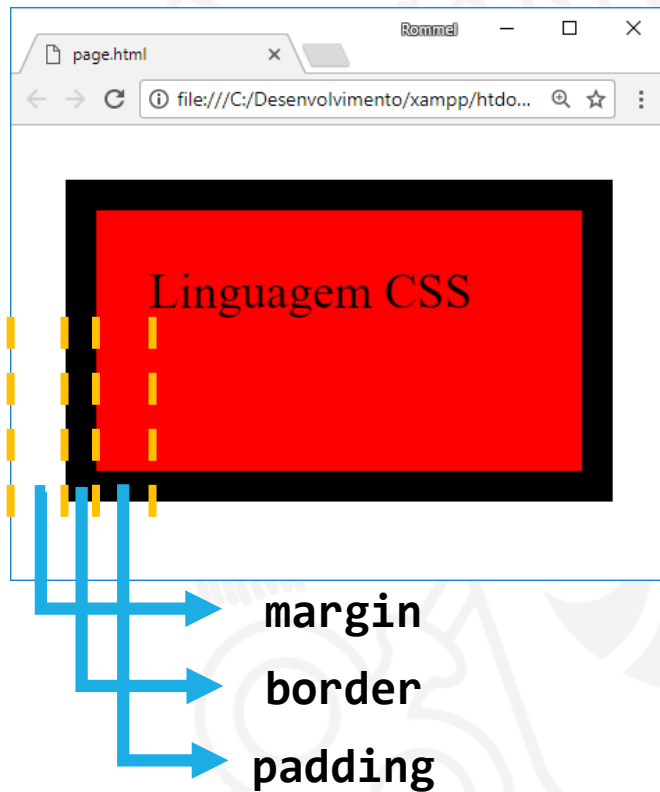
- **height** e **width**  
Tamanho básico do elemento
- **margin**  
Espaço ao redor do elemento
- **border**  
borda em torno do elemento
- **padding**  
Espaço entre a borda e o conteúdo
- **background** permite controlar:
  - cor de fundo do elemento
  - imagem como fundo



Fonte da Imagem: Mozilla Developer Network  
[https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction\\_to\\_CSS/Box\\_model](https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Box_model)

# CSS – Box Model

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    * { margin: 0px; padding: 0px; }
    div {
      height: 100px;
      background-color: red;
      margin: 35px;
      padding: 35px;
      border: solid 20px black;
    }
  </style>
</head>
<body>
  <div>Linguagem CSS</div>
</body>
</html>
```





# CSS – Box Model

## Forma de aplicação de tamanhos

- 1 valor: Aplica-se a todos

```
border: 10px;
```

**ABCD**

- 2 valores: o primeiro se aplica a top e bottom e o segundo se aplica a right e left

```
border: 5px 20px;
```

**AC**

**BD**

- 3 valores: o primeiro se aplica a top, o segundo a left e right e o terceiro a bottom

```
border: 5px 20px 10px;
```

**A**

**BD**

**C**

- 4 valores: Se aplicam na ordem do horário (top, right, bottom, left)

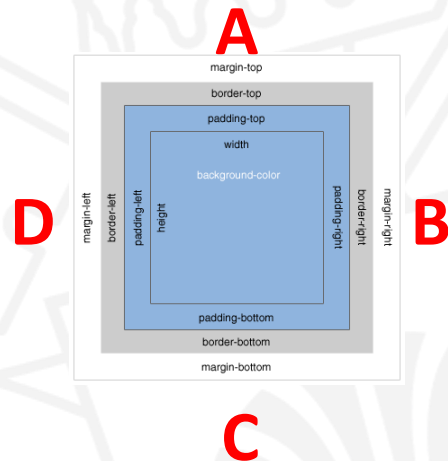
```
border: 5px 20px 10px 20px;
```

**A**

**B**

**C**

**D**



# CSS – Valores e Unidades – Comprimento e Tamanho

Tipo	Medida	Significado	Observação
Absoluto	in	Polegadas	Relação entre as medidas absolutas: 1in = 2,54cm = 25,4mm = 72pt = 6pc
	cm	Centímetros	
	mm	Milímetros	
	pt	Pontos	Padrão em tipografia, usados em impressoras. Existem 72 pontos para cada polegada
	pc	Paicas	Equivalente a 12 pontos
Relativo	em	Tamanho da Fonte	Por exemplo, 1.2em é o mesmo que 120%. Muito utilizado para margens.
	px	Pixels	
	%	Percentual	Por exemplo: H1 { line-height: 120% } seta a separação entre linhas para 120% do tamanho corrente de altura.

# CSS – Valores e Unidades – Cores

## Formas de especificar cores

- **#RRGGBB**  
Hexadecimal p/ vermelho (R), verde(G) e azul (B) que permite valores que variam de 0 (00) a 255 (FF).  
Ex: #FF88CC
- **#RGB**  
Uma forma curta para os valores em Hexa para com valores repetidos.  
Ex: #F8C é o mesmo que #FF88CC
- **rgb(rrr, ggg, bbb)**  
Especifica os valores de vermelho (R), verde(G) e azul (B) de 0 até 255  
Ex: verde puro rgb(0, 255, 0)
- **Palavras chaves**  
Palabras baseadas nas cores originais do Windows VGA  
Ex: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, e yellow.



**PUC Minas**  
**Virtual**

# Linguagem JavaScript

Rommel Vieira Carneiro

# Linguagem JavaScript

## Histórico e Poder da Linguagem

Rommel Vieira Carneiro

# O que é Javascript?

HTML define a estrutura do conteúdo

CSS define o formato de apresentação

Javascript adiciona a interatividade e permite a criação de aplicações ricas

# Histórico do JavaScript

- O **JavaScript** foi criado em 1995 por **Brendan Eich** na Netscape
- Em 1996, foi transferido para o ECMA International para padronização, sendo a primeira versão do ES lançada em 1997.
- Em 1998, a linguagem é aprovada como a **ISO/IEC 16262** e batizada então como **ECMA-262**
- Em 2005, o advento do **AJAX** alavancou o uso do JavaScript
- Em 2009 foi lançado o **ECMAScript 5 (ES 5)**, versão com maior compatibilidade nos browsers atualmente
- Em 2009 foi criado o projeto **CommonJS** com o objetivo de levar o JavaScript para além do Browser.
- Em 2015 foi lançado o **ECMAScript 6 (ES6)**, atualmente em estágio avançado de adoção pelos browsers
- O TC39 (<https://github.com/tc39>) é o Comitê Técnico responsável pela evolução do ECMAScript.

1995 ○ JS 1.0

1999 ○ ES 3

2009 ○ ES 5

2015 ○ ES 6

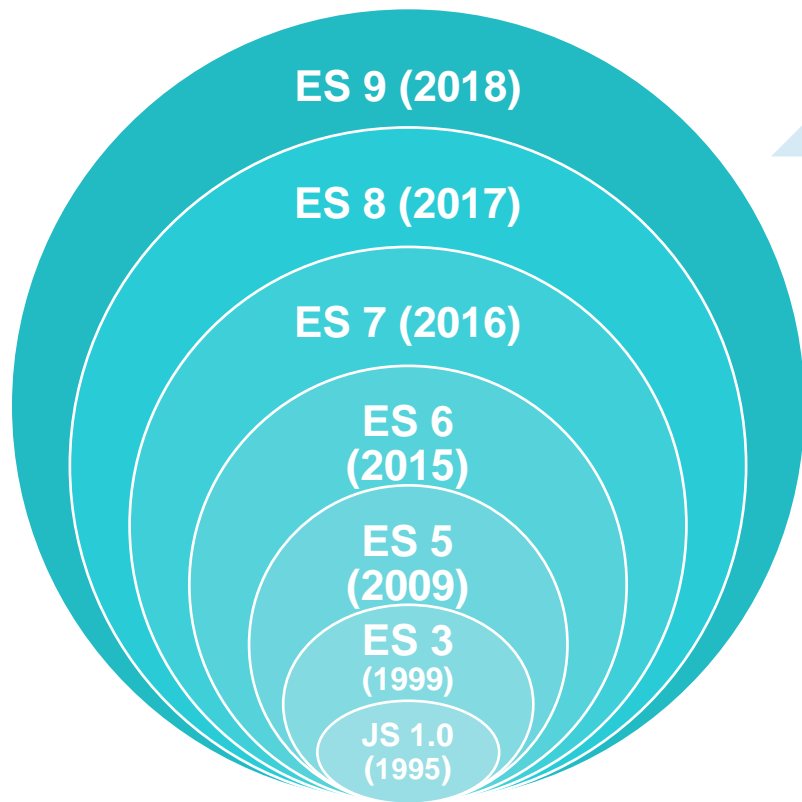
2016 ○ ES 7

2017 ○ ES 8

2018 ○ ES 9



# Histórico do JavaScript

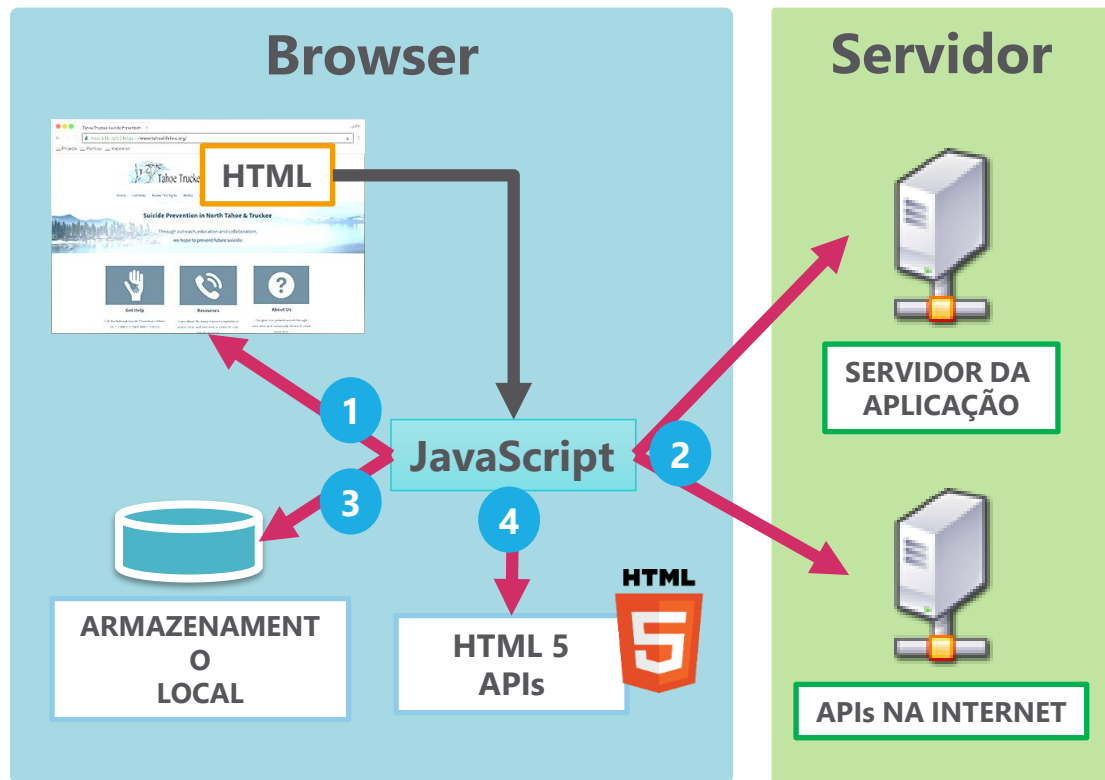


- **ECMA Script 9 (2018)**  
Regex Changes, Promise Finally, Async Iteration
- **ECMA Script 8 (2017)**  
Async functions, Shared Memory, Atomics
- **ECMA Script 7 (2016)**  
Array Includes, Exponential operator (\*\*)
- **ECMA Script 6 (2015)**  
Classes, Modules, Arrow Functions, Const e Let, Template Literals, Promises
- **ECMA Script 5**  
Strict mode, Suporte ao JSON
- **ECMA Script 3**  
Expressões regulares (RegExp), Métodos da classe String, Tratamento de exceção (try/catch)
- **JavaScript 1.0**

# Implementações do ECMAScript

Ambiente	Web Engine	ECMAScript Engine
Mozilla Firefox	Gecko	Spider Monkey <a href="https://developer.mozilla.org/pt-BR/docs/Mozilla/Projects/SpiderMonkey">https://developer.mozilla.org/pt-BR/docs/Mozilla/Projects/SpiderMonkey</a>
Google Chrome	Blink	Google V8 <a href="https://developers.google.com/v8/?hl=pt">https://developers.google.com/v8/?hl=pt</a>
Apple Safari	WebKit	JavaScriptCore <a href="https://developer.apple.com/documentation/javascriptcore">https://developer.apple.com/documentation/javascriptcore</a>
Internet Explorer	Trident	Chakra Core <a href="https://github.com/Microsoft/ChakraCore">https://github.com/Microsoft/ChakraCore</a>
Edge	EDGE	Chakra Core <a href="https://github.com/Microsoft/ChakraCore">https://github.com/Microsoft/ChakraCore</a>
		Rhino <a href="https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino">https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino</a>
Opera	Presto → Blink	

# JavaScript – Poder do JavaScript



- 1 Manipulação de objetos da página HTML e tratamento de eventos (**DOM Document Object Model**)
- 2 Comunicação com o servidor e uso de Application Programming Interface (API) via AJAX (**XMLHttpRequest**)
- 3 Persistência de dados em estruturas providas pelo Browser (**Indexed DB e LocalStorage**)
- 4 Interação com recursos das novas APIs do HTML 5 (**Canvas, Media, File, Drag and Drop, Geolocation, Web Workers, History**)

# JavaScript – Poder do JavaScript

- Aprimoramento do suporte a Orientação a Objetos
- Ambiente Node.js
- Desenvolvimento no desktop



# Linguagem JavaScript

## Conceitos Básicos da Linguagem

Rommel Vieira Carneiro

# Formas de Utilização do Javascript

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>Exemplo Javascript</title>
```

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

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

```
      /* código Javascript */
```

```
      alert ('Passei por aqui!');
```

```
    </script>
```

```
  </head>
```

```
  <body>
```

```
    <p onClick="alert('Voce clicou no parágrafo');">...</p>
```

```
  </body>
```

```
</html>
```

Arquivo JS externo

Bloco interno

Inline

# JavaScript – Questões básicas

## Gramática

- JavaScript é *case sensitive* e utiliza conjunto de caracteres **Unicode**  
**Nome** é diferente de **nome** que é diferente de **NOME**
- Comentários

```
// uma linha de comentário
```

```
/*  
    Comentário de  
    várias linhas  
*/
```

# JavaScript – Questões básicas

## Declarações de variáveis e constantes

- Declaração de variáveis, opcionalmente com atribuição de valor

```
var a = 0;
```

- Declaração de variáveis de escopo local, opcionalmente com atribuição de valor

```
let a = 0;
```

- Declaração de constantes

```
const A = 0;
```

- Valores constantes não podem ser alterados
- O escopo das constantes é semelhante ao das variáveis declaradas como let.

### Escopo de variáveis

```
<script>
  var a = 1;
  {
    let c = 5;
  }
  console.log (a);
  console.log (c);
</script>
```

---

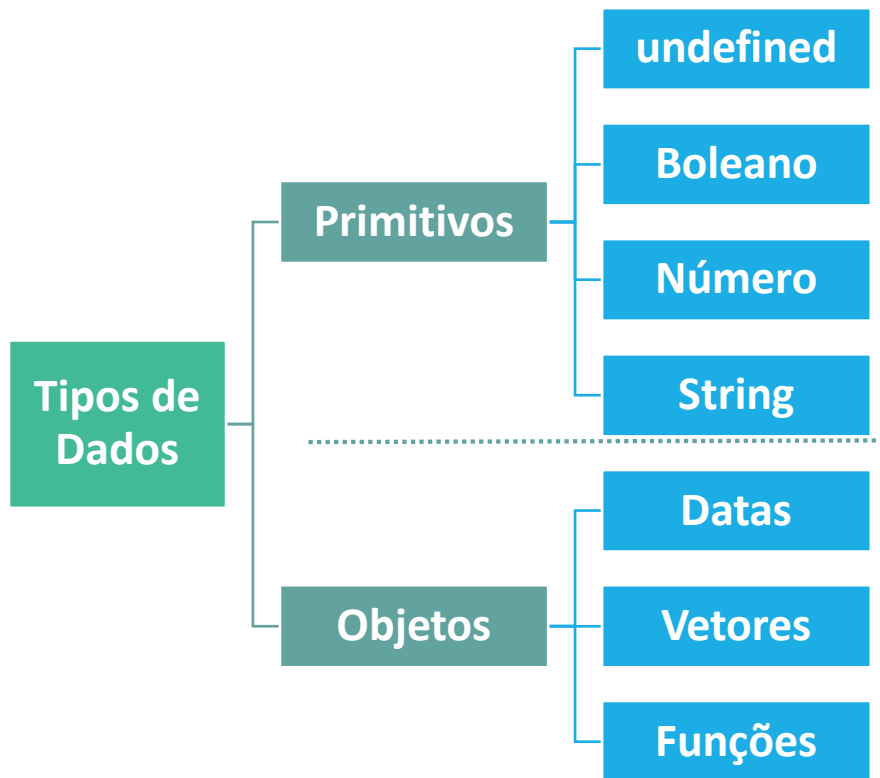
**Saída**

1

Uncaught ReferenceError: c is not defined



# JavaScript – Tipos e Estruturas de Dados



## Tipagem Dinâmica

```
<script>
  var a = 1;
  console.log (a);
  var a = 2.5;
  console.log (a);
  var a = "abc";
  console.log (a);
  var a = true;
  console.log (a);
  var a = ["a", "e", "i", "o", "u"];
  console.log (a);
</script>
```

**Saída**

```
1
2.5
abc
true
["a", "e", "i", "o", "u"]
```

# JavaScript – Tipos e Estruturas de Dados

## Booleano

- Variáveis booleanas podem ter valores **true** ou **false**
- Os operadores lógicos permitem comparações de expressões. São eles:

Operador	Descrição	Exemplo	Resultado
==	Igual a	x == y	true se x e y são iguais
===	Igual e mesmo tipo	x === y	true se x e y são iguais e do mesmo tipo
!==	Não igual a	x !== y	true se x e y são diferentes
>	Maior que	x > y	true se x é maior que y
<	Menor que	x < y	true se x é menor que y
>=	Maior ou igual a	x >= y	true se x é maior ou igual a y
<=	Menor ou igual a	x <= y	true se x é menor ou igual a y
!	Negativo	!x	true se x é false
&&	E exclusivo (ambos true)	x && y	true se x e y são true
	Ou (pelo menos um true)	x    y	true se ou x ou y são true

# JavaScript – Tipos e Estruturas de Dados

## Números

- Todos os números são de ponto flutuante com tamanho de 64 bits
- O objeto Math contém funções matemáticas avançadas
- NaN é o valor retornado por qualquer operação que não resulte em um número válido
- Existem os valores infinito e menos infinito.
  - Number.POSITIVE\_INFINITY e Number.NEGATIVE\_INFINITY

## Números

```
<script>
  var a = 123;
  console.log (a);
  var a = 1.23;
  console.log (a);
  var a = Math.PI;
  console.log (a);
  var a = 5.8;
  console.log (Math.round (a),
    Math.trunc(a), 0/0);
</script>
```

---

**Saída**

```
123
1.23
3.141592653589793
6 5 NaN
```

# JavaScript – Tipos e Estruturas de Dados

## Strings

- Strings podem ser declaradas com aspas ("abc") ou apostrofes ('abc')
- São compostas por sequências de caracteres Unicode
- O operador (+) pode ser utilizado para concatenar strings
- Embora sejam primitivos, é possível utilizar métodos e propriedades a partir de variáveis do tipo string
  - length → tamanho da string
  - charAt() → Caracter específico
  - toUpperCase() → String maiúscula
  - toLowerCase() → String minúscula

## Strings

```
<script>
  var a = "abc";
  console.log (a);
  var a = 'abc';
  console.log (a);
  console.log ("Tamanho de a: " +
               a.length);
  console.log (a.toUpperCase());
</script>
```

---

**Saída**

```
abc
abc
Tamanho de a: 3
ABC
```

# JavaScript – Tipos e Estruturas de Dados

## Strings – Caracteres Especiais

Para se representar caracteres especiais é necessário utilizar a notação de escape com a barra invertida (\).

### Exemplo

```
<script>
  var a = "Exemplo\nOlá \'Mundo\'";
  console.log (a);
</script>
```

Saída

```
Exemplo
Olá 'Mundo'
```

Código	Saída
\0	Caracter NULL
\'	Aspas simples
\"	Aspas duplas
\\	Barra invertida
\n	Nova linha (ENTER)
\r	Retorno de Carro (CR)
\v	Tabulação vertical
\t	Tabulação
\b	Backspace
\f	form feed
\uXXXX	Caracter Unicode
\xxx	Caracter Latin-1

# JavaScript – Tipos e Estruturas de Dados

## Template Strings

Strings definidas via (``) sinal de crase que podem interpretar expressões via construção `${ expression }` contidas na string.

### Template Strings

```
<script>
  var a = 5;
  var a = 10;
  console.log (a);
  console.log (`a + b = ${a + b} e \n
               a + b = ${a + b}.`);
</script>
```

---

**Saída**

```
a + b = 15 e
a * b = 50.
```

# JavaScript – Tipos e Estruturas de Dados

## Conversão de strings em números

- Função parseInt ()
- Função parseFloat ()

## Conversão de números para strings

- A conversão é automática

### Conversões

```
<script>
  var a = parseInt ("256");
  console.log (a);
  var a = parseInt ("2.56")
  console.log (a);
  var a = parseFloat ("2.56");
  console.log (a);
  var a = "valor: " + 2.56;
  console.log (a);
</script>
```

---

#### Saída

```
256
2
2.56
Valor: 2.56
```

# JavaScript – Tipos e Estruturas de Dados

## Objetos

- Em JavaScript, os objetos são coleções de pares nome e valor
  - O nome é uma string em JavaScript
  - O valor pode ser de qualquer tipo, incluindo outros objetos
- Os objetos são dinâmicos e podem receber novas propriedades e métodos a qualquer momento
- JSON ou **JavaScript Object Notation** é a forma como se descrevem os objetos em JavaScript

## Objetos

```
<script>
  var o = new Object();
  o.name = "João";
  o["age"] = 16;
  console.log (o.name+"-"+o.age);

  var email = {
    mensagem: "Olá Pamela",
    detalhes: {
      para: "Pamela",
      de: "João"
    }
  };
  console.log (email.mensagem);
</script>
```

---

**Saída**

João-16  
Olá Pamela



# JavaScript – Tipos e Estruturas de Dados

## Date

- As variáveis Date são objetos que permite tratar Datas e Horas
- Existem diversos construtores para Datas:
  - Date ()
  - Date (milissegundos)
  - Date (string)
  - Date (Ano, Mês, Dia, Hora, Minuto, segundo, milissegundo)
- Ao referenciar o número dos meses, é importante saber que janeiro é 0 e dezembro é 11.

## Objetos – Datas

```
<script>
  var a = new Date();
  console.log (a.toLocaleString());
  var a = new Date(2016, 0, 31);
  console.log (a.toLocaleString());
</script>
```

---

**Saída**

03/09/2016 15:07:58

31/01/2016 00:00:00

# JavaScript – Tipos e Estruturas de Dados

## Vetores (Arrays)

- Os vetores também são objetos
- A criação pode ser feita de duas formas
  - Função Array ()
  - Atribuição de literal
- Possuem a propriedade length que retorna o número de itens

## Objetos – Vetores

```
<script>
  var a = new Array();
  a[0] = "vermelho";
  a[1] = "preto";
  console.log (a);

  var b = new Array("rosa", "azul");
  console.log (b);

  var c = ["amarelo", "verde"];
  console.log (c);
  console.log (c.length);
</script>
```

---

**Saída**

```
["vermelho", "preto"]
["rosa", "azul"]
["amarelo", "verde"]
2
```

# JavaScript – Eventos

- O código JavaScript é executado a partir de atributos relacionados com eventos associados aos elementos do HTML.
- Os controles de evento são incluídos como atributos dos elementos HTML

```
<body onload="alert ('Página carregada.');">
```

No exemplo acima, assim que o corpo (**body**) for carregado (**onload**), será disparado o código

```
alert ('Página carregada.');
```

# JavaScript – Eventos

- Atualmente, a forma mais comum de lidar com os eventos de um objeto é por meio de código JavaScript que "injeta" esse tratamento fora do código HTML.
- Isso torna o código HTML mais legível separando o código de tratamento do JavaScript.
- Veja o exemplo:

```
<button id="btnHello">Hello</button>

-----

<script>
  var elem = document.getElementById('btnHello');
  elem.addEventListener('click', function () {
    alert("Hello World!");
  }, false);
</script>
```

# JavaScript – Eventos

Evento	Quando ocorre...	Controle
click	Usuário clica no link ou element de formulário	onclick
change	Usuário altera o valor de elemento de texto, textarea ou seleção	onchange
focus	Usuário posiciona o foco em um element de formulário	onfocus
blur	Usuário tira o foco de um element de formulário	onblur
mouseover	Usuário move o mouse sobre um elemento	onmouseover
mouseout	Usuário move o mouse para fora de um elemento	onmouseout
select	Usuário seleciona um campo de entrada de formulários	onselect
submit	Usuário submete um formulário	onsubmit
resize	Usuário redimensiona a janela do browser	onresize
load	O elemento é carregada no browser (window, body, frame, img, link)	onload
unload	Usuário sai da página	onunload

Esses são alguns exemplos de eventos. Existem outros.

# JavaScript – Eventos

## Exemplo

```
<!DOCTYPE html>
<html>
<head>
  <title>Exemplo DOM - Eventos</title>
</head>
<body>
  <p onmouseover="this.style.color='red';"
    onmouseout="this.style.color='black';" >
    Olá Mundo
  </p>
</body>
</html>
```

No exemplo acima, ao passar o mouse sobre o parágrafo, a cor do texto é alternada para vermelho.

A palavra-chave **this**, significa o próprio elemento onde o evento se encontra.

# JavaScript – Funções

- Sistema muito flexível - Todas as funções são objetos JavaScript
- Podem ter qualquer número de parâmetros
- Mais parâmetros podem ser passados do que especificados pela função
- Os parâmetros não são obrigatórios
- Podem retornar um valor explícito ou undefined

```
function soma( x, y ) {  
    var total = x + y;  
    return total;  
}  
soma()           // retorna NaN  
soma(2,3)        // retorna 5
```

# JavaScript – Funções – Exemplo

```
function media() {  
    var soma = 0;  
    for( var i=0, j=arguments.length; i<j; i++ ) {  
        soma += arguments[i];  
    }  
    return soma/arguments.length;  
}
```

media( 2, 3, 4, 5 ) retorna 3.5

media.apply( null, [2, 3, 4, 5] ) retorna 3.5



# Linguagem JavaScript

## AJAX – Asynchronous JavaScript and XML

Rommel Vieira Carneiro

# Introdução

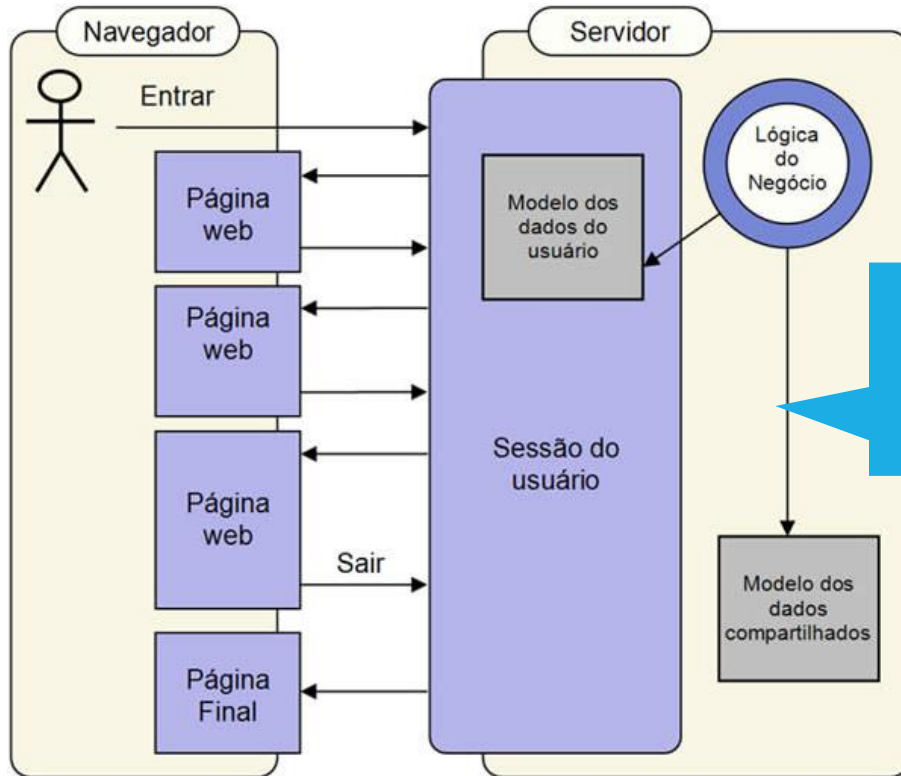
- Asynchronous Javascript and XML (AJAX) não é uma tecnologia, mas várias delas combinadas:
  - apresentação baseada em padrões como XHTML e CSS
  - exibição dinâmica através do DOM
  - troca e manipulação de dados com XML
  - recuperação assíncrona de dados com **XMLHttpRequest**
  - Javascript para juntar tudo isso
- Exemplo prático: GMAIL
  - Navegação entre pastas/tags
  - Carga automática de mensagens
  - Salvamento de mensagens durante a digitação
  - Sugestão de contatos



# Modelo Clássico vs Modelo AJAX

- No modelo clássico, um aplicação web trabalha assim:
  - O usuário dispara uma solicitação HTTP para o servidor web
  - O servidor web processa e então retorna uma página HTML
  - Repete-se esse processo indefinidamente durante a sessão do usuário
- No modelo AJAX:
  - O usuário dispara uma solicitação HTTP para o servidor Web
  - O servidor retorna uma aplicação (HTML + Javascript), e não apenas conteúdo HTML
  - A aplicação interage com o servidor, via Javascript, solicitando dados JSON/XML
  - O usuário interage de forma contínua sem a recarga constante da página web

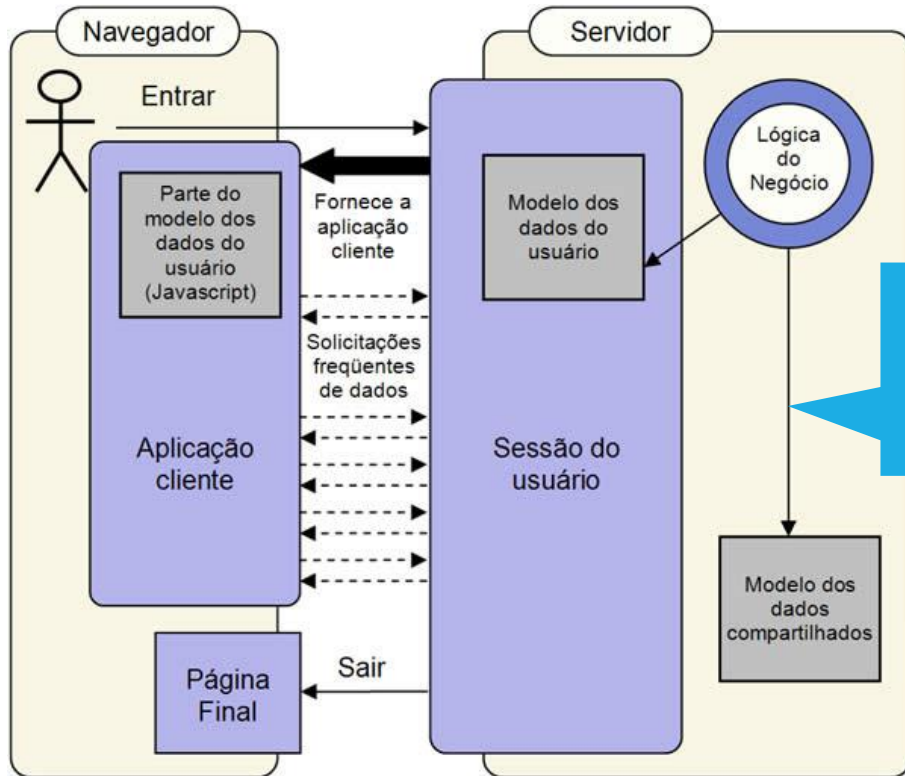
# Modelo Clássico vs Modelo AJAX



## Modelo Clássico

- Uma requisição → Uma página
- Lógica toda no servidor

# Modelo Clássico vs Modelo AJAX



## Modelo AJAX

- Diversas requisições para uma página
- Lógica dividida entre servidor e cliente

# Exemplo AJAX

```
<!DOCTYPE html>
<html>
  <head><script>
    function loadXMLDoc() {
      var xmlhttp = new XMLHttpRequest();

      xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4 && xmlhttp.status==200) {
          document.getElementById("secao").innerHTML = xmlhttp.responseText;
        }
      }
      xmlhttp.open("GET", "http://site.com/webservice.php", true);
      xmlhttp.send();
    }
  </script></head>
  <body>
    <div id="secao"><h2>Conteúdo a ser alterado</h2></div>
    <button type="button" onclick="loadXMLDoc()">Alterar</button>
  </body>
</html>
```

# Vantagens e Desvantagens

- Vantagens

- Melhoria significativa na experiência do usuário
- Redução do tráfego na rede
- Redução na carga do servidor web
- Flexibilidade de desenvolvimento do lado servidor

- Desvantagens

- Maior complexidade no desenvolvimento de aplicações
- Exigência um equipamento melhor no lado cliente
- Requer compatibilidade do browser com padrões Javascript, HTML e CSS
- Possui tratamento diferenciado de um browser web para outro
- Tratamento complexo para uso das opções de avançar e voltar do browser



# Objeto XMLHttpRequest

- Criado pela Microsoft e depois adotado pela Mozilla, o XMLHttpRequest é um objeto ou API, fornecida pelo Browser, que permite que programas Javascript troquem dados com servidores Web
- O objeto XMLHttpRequest é a base para a programação baseada em AJAX
- Apesar do nome, permite a troca de dados em outros formatos como JSON, HTML, TXT, entre outros, além de possibilitar a comunicação por meio de outros protocolos que não o HTTP.



# Objeto XMLHttpRequest – Propriedades

Propriedade	Descrição
status	O código de status HTTP da resposta da solicitação.
statusText	O texto de status HTTP que vai com o código.
readyState	O status do pedido.
responseText	Texto bruto da resposta.
responseXML	Resposta analisada em um objeto de documento DOM. Funciona apenas se o tipo de conteúdo é text/xml.
onreadystatechange	Disparo de evento chamado quando o readyState muda.
onerror	Disparo de evento que é chamado quando um erro acontece durante uma solicitação. (Somente Mozilla)
onprogress	Disparo de evento que é chamado em um intervalo, como o conteúdo é carregado. (Somente Mozilla)
onload	Disparo de evento que é chamado quando o documento for concluído. (Somente Mozilla)

# Objeto XMLHttpRequest – Fluxo ReadyState

Código de Status	Descrição
(0) UNINITIALIZED	O objeto foi criado mas não foi inicializado (O método open não foi disparado.)
(1) LOADING	O objeto foi criado, porém o método send não foi executado.
(2) LOADED	O método send foi executado, porém o status e os cabeçalhos ainda não estão disponíveis.
(3) INTERACTIVE	Alguns dados foram recebidos. Utilizar as propriedades responseBody e responseText neste estado para obter resultados parciais vai gerar um erro, uma vez que o status e os cabeçalhos ainda não estão completamente disponíveis.
(4) COMPLETED	Todos os dados foram recebidos e os dados completos estão disponíveis através das propriedades responseBody e responseText.



**PUC Minas**  
**Virtual**