

Disciplina

Tecnologia Web

Prof. Msc. Yuri Maximilian Rottner Dirickson

Formado em Bacharelado em Matemática Aplicada pelo Instituto de Matemática e Estatística da Universidade de São Paulo (IME-USP).
2012

Mestre em Ciências em Engenharia da Computação pela Escola politécnica da Universidade de São Paulo (EP-USP) 2016.

Analista de Sistemas pela Superintendência de Tecnologia da Informação da Universidade de São Paulo desde 2012. Responsável pela análise e desenvolvimento dos sistemas corporativos da universidade, atuando nas áreas de desenvolvimento frontend e backend.

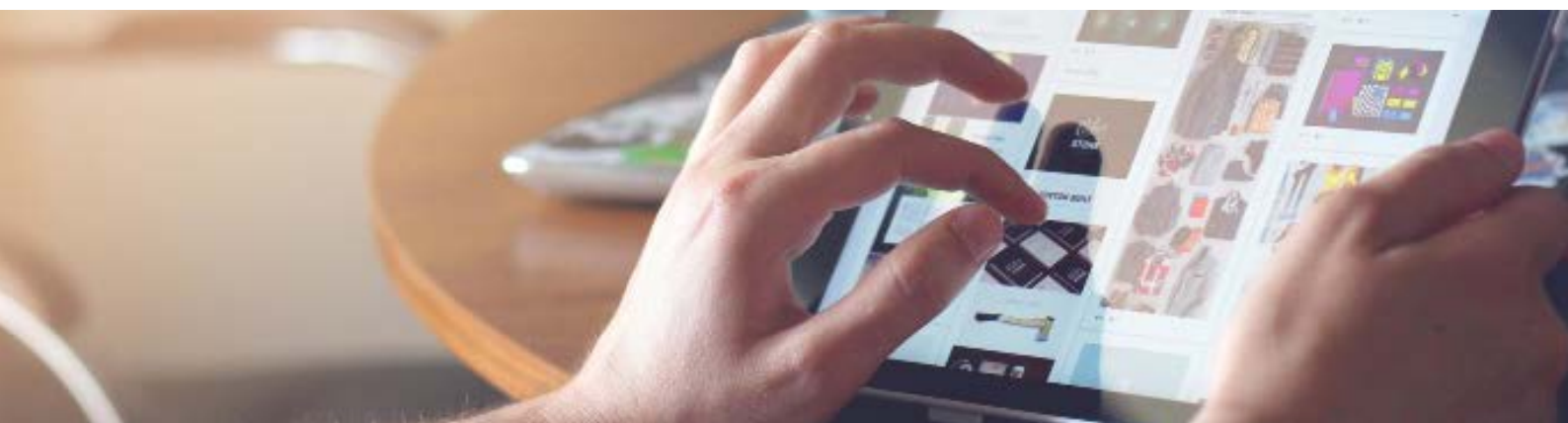
Professor adjunto pela Faculdade Impacta de Tecnologia desde 2017. Preparação de material, aula e condução das disciplinas: Tecnologias Web, Linguagem de Programação Python, Pesquisa e Ordenação de Dados (Java) e Estrutura de Dados (Java e Python).

DIRICKSON, YURI MAXIMILIAN ROTTNER; VISMARI, LUCIO FLAVIO; CAMARGO, JOAO BATISTA. Elicitation of Operational Requirements in ATC Activities Automation Process: The TWR Case. In: 2013 III Brazilian Symposium on Computing Systems Engineering (SBESC), 2013, Niterói. 2013 III Brazilian Symposium on Computing Systems Engineering, 2013. p. 149.



O professor

DIRICKSON, Y. M. R.; CAMARGO JR, J. B. ; VISMARI, L. F. . A METHODOLOGICAL APPROACH TO SAFETY ANALYSIS OF AUTOMATED TASKS OF TWR ATCO. In: XIII SITRAER? Air Transportation Symposium, 2014, São Paulo. Desafios em transporte aéreo e avaliação do transporte aéreo durante a Copa do Mundo FIFA 2014. São Paulo: JBATS, 2014. v. 1.



Introdução	4
Capítulo 1 - Protocolo HTTP	5
Capítulo 2 - Introdução a HTML5.....	12
Capítulo 3 - Formulário HTML5	22
Capítulo 4 - Introdução ao CSS6	30
Capítulo 5 - Layouts com CSS6.....	39
Capítulo 6 - Manipulando páginas com Java Script	46
Capítulo 7 - Padrão MVC e o Django Framework	55
Capítulo 8 - Estilo Capx	66
Capítulo 9 - Mapeamento de URL`s	69
Capítulo 10 - Conectando o BD no Django	74
Capítulo 11 - Autenticação a autorização Django	82
Capítulo 12 - Requisições AJAX	88

Introdução

Contextualização

Atualmente busca-se cada vez mais a automação de processos para se reduzir custos, aumentar produtividade e manter um nível alto de competitividade. Nesse contexto, a web despontou como a principal plataforma de transferência de informações.

Com isso é imprescindível a formação de profissionais que consigam analisar e projetar sistemas que lidem com essa plataforma de forma eficiente. Essa plataforma, que chamamos de cliente e servidor, abre um grande leque de opções para os profissionais: de interfaces inteligentes e reativas no frontend a serviços e sistemas robustos no backend.

Um profissional que, mesmo sendo especialista em uma dessas áreas, consiga transitar nos desafios que aparecem na aplicação dos conceitos se torna inestimável para uma empresa.

Formação Profissional

A disciplina de tecnologias cobre o conteúdo essencial para o estudante adentrar o mercado de trabalho de desenvolvimento web nas suas duas frentes: o frontend e o backend.

O aluno será capaz de projetar e desenvolver interfaces gráficas para a web com as tecnologias atuais (HTML5, CSS3 e JavaScript). Para o desenvolvimento no servidor, ele terá capacidade de aplicar o padrão MVC para o desenvolvimento de sistemas, aprendendo a entender o funcionamento de um framework de desenvolvimento, através do estudo com o Django Framework e a linguagem Python.

Capítulo 1

Protocolo HTTP

- Introdução;
- Modelo Cliente-Servidor
- Componentes do Cliente-Servidor
- Protocolo HTTP

1.1 Introdução

A plataforma web é uma rica em especificações, protocolos e padronizações. Por ser tão popular e utilizada no mundo inteiro, faz-se necessário toda essa padronização. No centro desse universo de siglas, entre IP, TCP, UDP, HTML, JS, CSS e etc. está o protocolo mais importante: o HTTP.

O HTTP padroniza todas as transferências de dados, arquivos, imagens e todo o resto que é transportado na web. Entender o seu funcionamento é essencial para o entendimento do desenvolvimento web.

1.2 Modelo Cliente-Servidor

Principal padrão arquitetural adotado na Internet. Processamento distribuído entre dois elementos.

- Cliente: requisita serviços.
- Servidor: realiza os serviços pedidos pelos clientes.

Exige comunicação entre os dois elementos:

- Necessidade de uma rede entre os computadores.
- Necessidade de um protocolo de comunicação.
- Necessidade de um mecanismo de localização.

Nessa estrutura temos os seguintes componentes:

- Navegador de Internet (Browser): Cliente, aquele que requisita algum recurso da internet.
- URL (<http://www.impacta.edu.br/>): endereço padronizado de recursos na Web (veremos adiante).
- Internet: uma sequência de redes de computadores toda interconectada.
- Servidor: computador (ou nuvem de computadores) que responde uma requisição feita por um cliente.

- Protocolos: Maneiras padronizadas de transmitir informações entre dois pontos em uma rede

O que vamos estudar na disciplina:

- Tecnologias que rodam no Navegador: HTML, CSS e JavaScript
- Estrutura básica de URLs.
- Programação para servidor (Python).
- Estrutura básica do HTTP

1.3 Componentes do Cliente-Servidor

A URL (Uniform Resource Locator) é um endereço para encontrar um determinado recurso na rede. Esse recurso pode ser qualquer coisa, como uma página php, imagem, vídeo, etc.

A rede em questão pode ser a internet, mas pode ser um recurso local da sua própria rede.

A URL é dividida em algumas partes, cada uma com a sua função.

- **Protocolo:** Chamado também de esquema, informa qual protocolo de comunicação será usado na requisição.
- **Domínio:** Indica o domínio da rede que hospeda o recurso desejada.
- **Path:** caminho interno até o recurso (chamado de path).
- **Resource:** recurso requisitado.

Para a web, temos basicamente dois atores: Cliente e Servidor. Cada um destes possui suas próprias tecnologias para operar. No cliente (ou frontend) temos:

- HTML(5)
- CSS(3)
- JavaScript

No servidor vamos ter:

- Servidores de Aplicação/HTTP (ou similar)
- Linguagens de Programação - Aplicação
- Bancos de Dados

1.3.1. HTML

Linguagem utilizada para representar o conteúdo e o formato visual de uma página no navegador. HTML é uma tagged language: suas tags definem a apresentação do conteúdo no navegador.

1.3.2. CSS

Padrão de formatação para páginas web que permite ao desenvolvedor ir além das limitações do HTML. Oferecem maior flexibilidade visual na programação de páginas HTML, gerando efeitos que somente poderiam ser conseguidos com imagens. Redução do tamanho das páginas HTML, com a redução do número de imagens necessárias no visual

1.3.3. JavaScript

Um script é uma sequência de comandos descritos em uma linguagem de programação para executar alguma ação. Os scripts de cliente são utilizados para dar “vida” ao HTML, ou seja, colocar comportamentos dentro das páginas (ex: ao clicar, abrir janela ou mostrar uma área que não existia).

Atualmente o ECMAScript é uma especificação e JavaScript é uma das implementações possíveis desses Scripts.

1.3.4. Servidor Web

Para que o computador que receberá a requisição seja capaz de interpretá-la, ele precisa entender o protocolo HTTP. Para entender esse protocolo, é necessário um software conhecido como servidor HTTP (ou web server).

1.3.5. Programação no Servidor

Para executar a lógica associada da sua aplicação, é necessária uma linguagem de programação que o seu servidor (computador) possa executar. Diversos tipos (compilado x interpretado), diversos objetivos (orientação a objeto, estrutural, funcional, aspectos), diversas preferências.

1.3.6. Sistema Gerenciador de Banco de Dados

Software utilizado para gerenciar bases de dados. Cuida do acesso, manutenção e persistência de dados nas suas bases. Garante o acesso a elas através de uma série de drivers e de uma linguagem específica de consultas e alterações (SQL).

1.4. Protocolo HTTP

É um protocolo de troca de informações na internet. Versão atual é a 1.1 (RFC 2616 em 1999), mas a versão HTTP2 já é uma realidade em muitos provedores e servidores. Utilizado sobretudo para navegação na internet, mas também por integrações de sistemas via web (REST).

É composto de requisições e respostas entre um cliente (um navegador na maioria dos casos) e um servidor.

Uma requisição é um pedido de um determinado recurso. Esse pedido vai com a sua especificação em termos de cabeçalhos de requisição (request headers) e com um identificador (URL) do recurso. Para que o servidor saiba o que deve fazer com essa requisição, o HTTP manda também um verbo, indicando qual ação executada pelo servidor.

É **stateless** (sem estado): uma requisição tem o ciclo de vida baseada na resposta. Após o retorno do servidor todas as informações sobre a requisição são perdidas. Para persistir informações entre requisições, precisamos usar recursos específicos:

- **Cookies:** Conjunto de pares chave-valor em texto que ficam armazenados no cliente e são enviados em cada requisição de um mesmo domínio.
- **Sessão:** Objeto em memória guardando dados específicos sobre a origem da requisição (dentro do servidor)

1.4.1. Métodos HTTP

Verbos ou Métodos HTTP indicam o que o servidor deve fazer ao receber a requisição, os mais usados são:

- GET: Usado para obter um recurso qualquer do servidor.
- POST: Envia dados para serem processados pelo servidor, em geral para criar ou alterar um recurso.
- DELETE: Remove um determinado recurso do servidor.
- PUT: Atualiza ou cria um recurso no servidor.
- HEAD: Obtém informações sobre um recurso, mas não o devolve.

1.4.2. Status HTTP

Após a requisição, o servidor processa toda a informação necessária e formata a resposta para enviar ao cliente. Essa resposta é formada por:

- cabeçalhos de resposta (response headers): metadados da resposta.
- status: código informando sobre a situação da resposta (se deu ou não certo).
- Corpo da resposta: o corpo contém informações disponibilizadas ao cliente, podendo ser o recurso pedido ou apenas uma confirmação de execução.
- O código de status é um número que varia de 100 a 600, cada um associado com um tipo de situação que pode ser encontrado durante a navegação na web.

Alguns dos mais comuns:

- 200 (Ok): Tudo deu certo.

- 403 (Forbidden): Você não possui permissão para este recurso (login).
- 404 (Not Found): Recurso não foi encontrado.
- 405 (Method not Allowed): Método HTTP é bloqueado pelo servidor.
- 500 (Internal Server Error): Algum problema interno do servidor (bug).

Headers	Preview	Response	Cookies	Timing
General Request URL: <code>http://www.impacta.edu.br/</code> Request Method: <code>GET</code> Status Code: ● 200 OK Remote Address: <code>189.39.8.7:80</code> Referrer Policy: <code>no-referrer-when-downgrade</code>				
Response Headers (12)				
Request Headers view source Accept: <code>text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8</code> Accept-Encoding: <code>gzip, deflate</code> Accept-Language: <code>pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4</code> Cache-Control: <code>no-cache</code> Connection: <code>keep-alive</code> Host: <code>www.impacta.edu.br</code> Pragma: <code>no-cache</code> Upgrade-Insecure-Requests: <code>1</code> User-Agent: <code>Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36</code>				

1.1 Exemplo de Requisição HTTP

Response Headers view source Cache-Control: <code>no-store, no-cache, must-revalidate, post-check=0, pre-check=0</code> Content-Length: <code>32011</code> Content-Type: <code>text/html; charset=UTF-8</code> Date: <code>Mon, 14 Aug 2017 20:05:15 GMT</code> Expires: <code>Thu, 19 Nov 1981 08:52:00 GMT</code> Link: <code><http://www.impacta.edu.br/wp-json/>; rel="https://api.w.org/"</code> Link: <code><http://wp.me/89GCP>; rel=shortlink</code> Pragma: <code>no-cache</code> Server: <code>Microsoft-IIS/7.5</code> Set-Cookie: <code>PHPSESSID=15eigm4lmb46cgk5r7j9ggmn91; path=/</code> X-Powered-By: <code>ASP.NET</code> X-Powered-By: <code>PHP/5.3.24</code>

1.2 Exemplo de Resposta HTTP

Pontos Principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O modelo que representa a arquitetura da web é o cliente – servidor. Nesse modelo é necessário alguém requisitando uma informação (cliente) e alguém respondendo a essa requisição (servidor).
- Diversos componentes estão por trás desse modelo cliente – servidor: HTML, CSS e JavaScript no cliente e um servidor web, linguagem de programação e um banco de dados no servidor.
- Entre todas essas tecnologias está o protocolo HTTP, ele conecta todas essas camadas com o transporte da informação necessária.

Capítulo 2

Introdução ao HTML5

- IO que é o HTML
- Como funciona o HTML;
- Estrutura básica;
- Tipos de Marcadores.

2.1. O que é HTML ?

HTML é um acrônimo para HyperText Markup Language, ou Linguagem de Marcação de HiperTexto em português. Hipertexto é um texto com características extras, como formatação, imagens, multimídia e links para outros documentos.

Marcação (ou Markup) é o processo de formatar o texto e adicionar símbolos extras. Cada símbolo usado pela marcação no HTML é um comando que diz ao navegador (browser) como exibir o texto.

Criada em 1990, por Tim Berners-Lee, logo após a criação da web. A web, em sua concepção original, foi definida como uma rede de documentos hipertexto. O próprio protocolo de comunicação na web, HTTP, é baseado na troca de hipertextos (acrônimo para HiperText Transfer Protocol). O HTML é um padrão definido pela W3C, com diversas versões existentes, sendo a mais nova o HTML5

O HTML5 combina novos elementos, dando mais recursos para a formatação das páginas. Além de recursos, as marcações adicionam maior valor semântico às páginas. Apesar do número 5 ser usado na marca, essa especificação está sempre em evolução:

- Versão 5.1 já lançada.
- Versão 5.2 em trabalho.

Cada versão traz mais poder em recursos e semântica as páginas da web. O HTML é uma linguagem de marcação, e não de programação. HTML não possui estruturas comuns em linguagens de programação como estruturas de seleção, estruturas de repetição, variáveis, métodos ou classes.

2.2 Como funciona o HTML?

É escrita em forma de marcadores (conhecidos como tags), criado pelos sinais < e >. Descreve a estrutura, aparência e semântica de um documento e permite encapsular códigos de linguagens de script. Um arquivo HTML é um arquivo de texto com uma extensão específica (.html ou .htm)

O navegador, ao receber um arquivo HTML do servidor, interpreta este arquivo e apresenta a interface gráfica para o usuário. Um arquivo HTML puro é composto basicamente de uma estrutura de marcadores aninhados e seus atributos.

Os marcadores dizem ao browser como o texto, a informação e as imagens serão exibidas. Por exemplo, um marcador diz se o texto será exibido em negrito, itálico e com um tipo de fonte qualquer.

Além de formatação de texto um marcador também pode dizer que o pedaço do texto é endereço de outra página Web (link).

Um marcador começa com “<” e termina com “>”: **<nome_do_marcador>**

Entre os sinais < > o nome do marcador. A maioria dos marcadores possui um formato de abertura e um formato de fechamento. A única diferença entre eles é que no marcador de fechamento existe uma barra (/) logo após o sinal de >

O formato genérico de um marcador é:

- **<marcador> Texto </marcador>**

Alguns marcadores não necessitam de fechamento, então assumem o seguinte formato:

- **<marcador />**

Os atributos definem propriedades dos marcadores, como tamanho, nome, identificador, classes CSS e valores:

- **<marcador atributo='valor'> Texto </marcador>**

Existem atributos comuns a vários marcadores e atributos que são específicos de um marcador. O HTML não é case sensitive, ou seja, não diferencia letras maiúsculas de minúsculas. Entretanto, a forma correta e padronizada é utilizar apenas letra minúsculas. Criem este hábito!

Aprender HTML é aprender os marcadores definidos no padrão e como utilizá-los para construir uma página Web.

O navegador não interpreta caracteres de quebra de linha (“enter”), tab, e mais que um caractere de espaçamento, ou seja, os três exemplos abaixo serão interpretados da mesma forma pelo navegador, exibindo o texto como no primeiro quadro

Formatação de texto

Formatação de texto

Formatação

de texto

2.1. Exemplos de caracteres em branco no HTML

Formatação de texto

Formatação de

texto

Formatação de texto

Formatação de texto

Formatação

de texto

Formatação de texto

2.2. Resultado de caracteres em branco no HTML

2.3. Estrutura Básica de um HTML

Agora vamos apresentar a estrutura básica de tags esperada de um documento HTML.

2.3.1. DOCTYPE

Indica que o arquivo é um documento HTML, devendo ser a primeira linha do arquivo.

Até a versão 4.01 do HTML, era necessário indicar a versão do HTML neste marcador. A partir da versão 5, esta declaração ficou simplificada. Como os navegadores modernos já estão preparados para interpretar a versão 5 do HTML, não é necessário declarar a versão:

```
<!DOCTYPE html>
```

2.3.3. HEAD

Delimita a área de cabeçalho. O cabeçalho engloba marcadores de metadados, scripts, folhas de estilo, título da página. Define os metadados da página: Informações adicionais sobre a página HTML como codificação de caracteres, autor do documento, idioma, palavras chave.

Importante para indexação das páginas pelos buscadores.

2.3.4. BODY

Delimita a área do corpo do documento. O corpo contém todas as tags e informações que devem ser mostradas ao usuário.

2.4. Tipos de Marcadores HTML

Vamos apresentar os marcadores mais utilizados e comuns do HTML5. Vamos dividir os marcadores nas seguintes categorias:

- Marcadores de Texto.
- Marcadores de Multimídia.
- Marcadores de Estrutura.
- Marcadores de Divisão.

2.4.1. Texto – Parágrafo

Define um parágrafo de texto. Cada texto é exibido em uma linha no navegador, definindo um espaçamento acima da primeira linha e abaixo da última linha.


```
<p> Parágrafo de texto </p>  
<p> Outro parágrafo </p>
```

Parágrafo de texto

Outro parágrafo

2.1. Exemplo de um parágrafo.

2.4.2. Texto Negrito e Itálico

```
<p>Parágrafo de texto <em>itálico</em></p>  
<p>Parágrafo de texto <strong>negrito</strong></p>
```

Parágrafo de texto *itálico*

Parágrafo de texto **negrito**

2.1. Exemplo de negrito e itálico.

2.4.3. Texto – Quebra de Linha

Define uma quebra de linha. O texto após `
` é exibido na próxima linha `
`. É um dos marcadores em que a abertura e fechamento estão apenas em um elemento (auto-contidos)

```
<p>Parágrafo de texto <em>itálico</em></p>  
<p>Parágrafo de texto <strong>negrito</strong></p>
```

Parágrafo de texto *itálico*

Parágrafo de texto **negrito**

2.1. Exemplo de negrito quebra de linha

2.4.4. Texto – Texto – Títulos

Os marcadores de <h1> </h1> até <h6> </h6> definem vários níveis de título dentro do corpo do HTML. Cada nível possui uma formatação diferente.

```
<h1>Título nível 1</h1> <h2>Título nível 2</h2>  
<h3>Título nível 3</h3> <h4>Título nível 4</h4>  
<h5>Título nível 5</h5> <h6>Título nível 6</h6>
```

Título nível 1

Título nível 2

Título nível 3

Título nível 4

Título nível 5

Título nível 6

2.1.Exemplo de títulos

2.4.5. Texto – ncoras

Reconhecido em uma página por estar em cor diferente do resto do texto e, ao passar com o mouse sobre esse texto o cursor muda para uma mão apontando para o link.

Para funcionar, o marcador <a> deve ser conter um atributo, href, que indica a URL destino ao clicar no link. O link pode ser para um recurso interno, do próprio website, ou uma referência para um website externo. O link pode referenciar qualquer recurso na web (html, arquivos, imagens).

Referência a um recurso

```
<a href='http://www.impacta.edu.br' > externo </a>
```

```
<br>
```

Referência a um recurso

```
<a href='pagina.html' > próprio </a>
```

Referência a um recurso [externo](#)

Referência a um recurso [próprio](#)

Já vimos que um link pode conter uma referência a um recurso externo ou a um recurso do próprio web site. Uma terceira aplicação do link é fazer referência a uma região da página web. A região é referenciada pelo link a través de seu id, utilizando o caractere “#”. Ao clicar no link, a página web “rola” até a região referenciada.

2.5. Marcadores de Mídia

Marcadores de mídia tem como efeito adicionar propriedades visuais e/ou sonoras para o website. Os marcadores são de três tipos: imagem, áudio e vídeo.

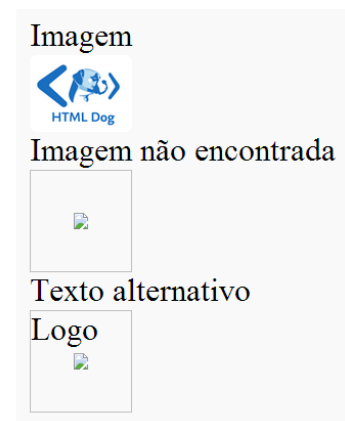
2.5.1. Mídia – Imagem

O marcador `` define uma região da página que contém uma imagem. Este marcador deve conter o atributo `src`, que indica a origem da imagem `` não tem marcador de fechamento

```
Imagem <br />
<img src='http://www.htmldog.com/badge1.gif'
width='50px' /><br />

Imagem não encontrada <br />
<img src='local_da_imagem.jpg' width='50px' />
<br/>

Texto alternativo <br />
<img src='local_da_imagem.jpg' alt='Logo'
width='50px' />
```



2.1.Exemplo de imagens

2.5.2. Mídia – Áudio e Vídeo

Ambos marcadores funcionam de maneira similar: definem uma região da página que conterá player de áudio/vídeo. Este marcador deve conter outras tags source, que indica a origem da mídia. Cada `<source>` indica um arquivo que ele deve procurar, caso o anterior não funcione.

O arquivo de som ou vídeo pode estar armazenada no próprio site ou fazer referência a um outro arquivo da web.

2.6. Marcadores de Estrutura

São marcadores utilizados para separar alguma estrutura textual, como tabelas e listas.

2.6.1. Estrutura – Listas Ordenadas e Não Ordenadas

` ` define uma lista não ordenada (ul = unordered list). ` ` define um item da lista. Item são precedidos por um “ponto”. A lista inicia sempre em uma nova linha.

Exemplo de lista comum

```
<ul>  
  <li> Item um </li>  
  <li> Item dois </li>  
</ul>
```

Exemplo de lista comum

- Item um
- Item dois

2.1.Exemplo de lista não ordenada

`` `` define uma lista não ordenada (ol = ordered list). `` funciona da mesma forma que a ``, apenas mudando os pontos da lista para números (ou letras).

Exemplo de lista ordenada

```
<ol>  
  <li> Item um </li>  
  <li> Item dois </li>  
</ol>
```

Exemplo de lista ordenada

1. Item um
2. Item dois

2.2.Exemplo de lista ordenada

2.6.2. Estrutura – Tabelas

`<table> ... </table>` Delimita uma tabela, `<tr> ... </tr>` delimita uma linha da tabela, `<td> ... </td>` delimita uma célula da tabela, devendo aparecer dentro de `<tr>`. `<th> ... </th>` delimita uma célula de título da tabela, devendo aparecer dentro de `<tr>`.

```
<table>
  <tr>
    <th>Cabeçalho 1</th>
    <th>Cabeçalho 2</th>
  </tr>
  <tr>
    <td>Cel 1-1</td>
    <td>Cel 1-2</td>
  </tr>
  <tr>
    <td>Cel 2-1</td>
    <td>Cel 2-2</td>
  </tr>
</table>
```

Cabeçalho 1 Cabeçalho 2

Cel 1-1 Cel 1-2

Cel 2-1 Cel 2-2

2.1.Exemplo de tabela

Pontos Principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- A Estrutura básica de um documento HTML não é estritamente necessário para o funcionamento da página, mas garante uma maior qualidade do seu documento;
- Tags podem ser escritas de duas maneiras: marcando um texto (ex: `<p>texto</tag>`) ou apenas para marcar um efeito no texto (ex: `
`);
- Treine e consulte as tags mais específicas para cada aplicação. Existem muitas tags para as situações mais diversificadas.

Capítulo 3

Formulário HTML5

- Introdução;
- Formulário;
- Campos de formulário;
- Etiquetas

3.1. Introdução

Até agora só temos uma maneira para coletar informações do usuário: a barra de endereços do navegador. Hoje quando pensamos em web, pensamos na quantidade de interações e informações que colocamos em diversos formulários que preenchemos: cadastros, transferências bancárias, provas, etc.

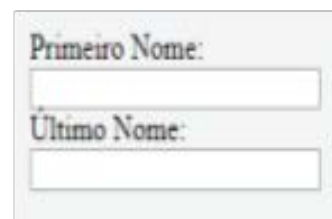
O formulário, e seus subcomponentes, é responsável por coletar informações das mais diversas do usuário e manda-las para o servidor.

3.2. Formulário

Por enquanto, a única informação que conseguimos coletar do nosso usuário é o recurso que ele deseja alcançar (via URL). Como podemos fazer para coletar outras informações, como por exemplo, um cadastro de cliente em uma loja?

Para isso o HTML possui um elemento chamado de formulário. Representado pela tag `form`, esta tag se comporta como uma `DIV` visualmente. Um formulário é composto basicamente de campos de coleta de dados (chamamos de `inputs`) e de etiquetas desses campos (chamamos de `labels`).

```
<form>
  <label for="primeiroNome">Primeiro Nome</label>:<br>
  <input type="text" name="primeiroNome" id="primeiroNome"><br>
  <label for="ultimoNome">Último Nome</label>:<br>
  <input type="text" name="ultimoNome" id="ultimoNome">
</form>
```



3.1 Exemplo de formulário com campos

Além dos campos, um formulário deve ter um botão de submissão (vamos ver adiante). Podemos também dividir os campos do nosso formulário em categorias (ex: dados pessoais e dados de entrega), usando o fieldset.

Para usarmos o form precisamos definir as seguintes propriedades:

- action: para onde o formulário deve enviar os dados.
- method: como o formulário deve enviar os dados.
- enctype: como o formulário deve empacotar os dados.
- name: um nome para identificar o formulário no JS.

3.2.1. Ação de Formulário

URL do servidor que irá receber esses dados para processamento. Pode ser URL absoluta (<https://SERVIDOR/novoCliente>) ou apenas relativa se for o mesmo servidor (</novoCliente> - preferível).

```
<form action="novoCliente">
  <label for="primeiroNome">Primeiro Nome</label>:<br>
  <input type="text" name="primeiroNome" id="primeiroNome"><br>
  <label for="ultimoNome">Último Nome</label>:<br>
  <input type="text" name="ultimoNome" id="ultimoNome">
</form>
```

3.2 Formulário com action

3.2.2. Método de Formulário

Define o método HTTP utilizado para enviar os dados. Os valores possíveis são GET e POST, sendo o POST o mais adequado.


```
<form action="novoCliente" method="POST">
  <label for="primeiroNome">Primeiro Nome</label>:<br>
  <input type="text" name="primeiroNome" id="primeiroNome"><br>
  <label for="ultimoNome">Último Nome</label>:<br>
  <input type="text" name="ultimoNome" id="ultimoNome">
</form>
```

3.3 Formulário com método

3.2.3. Codificação de Formulário

Em geral, o formulário empacota os dados usando a codificação de URL (application/x-www-form-urlencoded). Essa maneira já converte caracteres diferentes antes de enviar.

Existe a opção multipart/form-data onde essa conversão não ocorre. Usamos essa opção quando há arquivos no formulário

3.3 Campos de Formulário

O HTML5 apresentou vários novos campos a serem usados. Vamos ver primeiro os principais a serem utilizados. Praticamente todo campo no HTML é identificado pela tag input. Essa tag não possui tag de fechamento: <input name="nome" />

Os campos do HTML podem ser separados em:

- Coleta de texto (com exceção do tipo arquivo!).
- Coleta de opções.
- Botões de controle.

Os inputs possuem vários atributos, alguns que dependem do tipo inclusive. Vamos apresentar os mais importantes.

São dois atributos obrigatórios: name e type

- type: indica o tipo do atributo (ex: text ou checkbox)
- name: indica o nome do atributo que o conteúdo do input irá enviar. Por exemplo, no campo com nome primeiroNome, se digitarmos Yuri e enviarmos o formulário, o corpo da requisição vai como primeiroNome=Yuri.

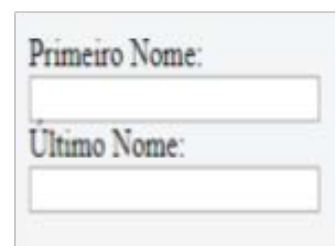
Outras propriedades:

- value: indica um valor inicial ao campo, ou a uma seleção.
- readonly: indica que o campo está como leitura apenas.
- disabled: indica que o campo não pode ser usado/enviado.
- size: indica o tamanho do campo em caracteres (texto).
- maxlength: indica o limite de caracteres que o campo permite (texto).
- min e max: indica os valores máximo e mínimo do campo (número e data)
- multiple: indica que pode escolher mais de um (seleção).
- placeholder: coloca um texto explicativo dentro do campo.
- required: indica que o campo é obrigatório (validação).

3.3.1 Campo de Texto

Tipo padrão de campo, feito para coletar textos de uma única linha. Quando o input não declarar seu tipo, ele será um text.

```
<form>  
  <label for="primeiroNome">Primeiro Nome</label>:<br>  
  <input type="text" name="primeiroNome" id="primeiroNome"><br>  
  <label for="ultimoNome">Último Nome</label>:<br>  
  <input type="text" name="ultimoNome" id="ultimoNome">  
</form>
```

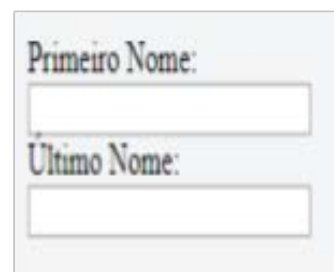


3. 4 Campo de texto

3.3.2 Campo de Senha

Similar ao text, mas esconde o que está sendo digitado

```
<form>  
  <label for="primeiroNome">Primeiro Nome</label>:<br>  
  <input type="text" name="primeiroNome" id="primeiroNome"><br>  
  <label for="ultimoNome">Último Nome</label>:<br>  
  <input type="text" name="ultimoNome" id="ultimoNome">  
</form>
```



3. 5 Campo de senha

3.3.3 Campo de Área de Texto

Essa tag define uma área de texto para ser escritas várias linhas de texto. Nesse caso, a tag possui abertura e fechamento.

```
<form>
  <label for="primeiroNome">Primeiro Nome</label><br>
  <input type="text" name="primeiroNome" id="primeiroNome"><br>
  <label for="ultimoNome">Último Nome</label><br>
  <input type="text" name="ultimoNome" id="ultimoNome">
</form>
```

3.4 Campo de texto

3.3.4. Campo de Seleção de Opções

Essa tag define uma caixa de seleção para o usuário. As opções são escritas diretamente dentro do elemento. A tag possui abertura e fechamento: `<select></select>`

Dentro da tag vai a lista de opções que o usuário terá disponível. Cada opção tem que estar dentro da tag `<option></option>`. A tag option possui um atributo com o valor dela (value), caso seja diferente do texto da opção.

```
<form>
  Estado:<br/>
  <select name="estado">
    <option value="SC">Santa Catarina</option>
    <option value="SE">Sergipe</option>
    <option value="SP">São Paulo</option>
  </select>
</form>
```

3.7 Caixa de Seleção de Opções

3.3.5. Caixa de Marcação de Opções

Ao invés de usar o select, o checkbox deixa todas as opções listadas na tela. O agrupamento de opções é feito pelo name dos inputs. O que é enviado para cada checkbox é o que estiver dentro da propriedade value.

Caso mais de um checkbox de um mesmo name seja selecionado, é enviado um vetor de valores para o servidor. O checkbox mesmo depois de selecionado pode ser “des-selecionado”.

```

<form>
  Linguagens:<br/>
  <input type="checkbox" value="Python" name="lp"/> Python<br/>
  <input type="checkbox" value="Java" name="lp"/> Java<br/>
  <input type="checkbox" value="Ruby" name="lp"/> Ruby<br/>
  Deseja receber novidades das linguagens selecionadas?
  <input type="checkbox" value="sim" name="novidade"/>
</form>

```

Linguagens:

☐ Python

☐ Java

☐ Ruby

Deseja receber novidades das linguagens selecionadas? ☐

3.8 Caixa de Marcação de Opções

3.3.6. Caixa de Marcação de Opção Única

Funciona de maneira similar ao checkbox, mostrando todas as opções, com a diferença de que apenas um pode ser selecionado (no mesmo grupo).

O agrupamento de opções é feito pelo name dos inputs. O que é enviado para cada radio é o que estiver dentro da propriedade value (padrão é on). O radio não pode ser “de-selecionado”.

```

<form>
  Midia Social:<br/>
  <input type="radio" value="face" name="midia"/> Facebook<br/>
  <input type="radio" value="twitter" name="midia"/> Twitter<br/>
  <input type="radio" value="gp" name="midia"/> Google+<br/>
  E-Mail:<br/>
  <input type="radio" value="gmail" name="email"/> GMail<br/>
  <input type="radio" value="hotmail" name="email"/> Hotmail<br/>
</form>

```

Midia Social:

☐ Facebook

☐ Twitter

☒ Google+

E-Mail:

☒ GMail

☐ Hotmail

3.9 Caixa de Marcação de Opção Única

3.3.7. Outros campos

Todos são variações do campo texto, inclusive quando o navegador não suportar o campo especificado, ele será renderizado como um campo texto comum.

Alguns desses novos campos são:

- email: campo texto que valida um e-mail.
- number: campo que aceita apenas números.
- url: campo texto que valida uma URL.
- datetime: campo que valida uma data com hora.
- color: campo texto que valida uma cor (hexadecimal).

Além da função semântica, esses novos campos também introduzem novos controles em navegadores com suporte (ex: datepicker ou colorpicker).

3.4. Etiquetas

É comum termos perto de cada campo um texto do seu significado (ex: a palavra E-Mail perto do campo de e-mail). Palavra soltas perto de campos não são automaticamente associados a eles, para que isso ocorra vamos usar a tag label.

Essa tag label serve apenas para conectar um texto de significado para um input qualquer (e apenas input). Para fazer essa conexão, usamos o atributo for do label ser igual ao atributo id do input.

Outro efeito do label associado a um campo é que ao clicar no label, ele passa o foco para o campo associado, facilitando a acessibilidade do seu site (principalmente para checkbox's e radios).

```
<form>
  <label for="input_email">E-Mail</label><br/>
  <input type="email" name="email" id="input_email"/>
  <br/>
  <label for="input_nome">Nome</label><br/>
  <input name="nome" id="input_nome"/><br/>
  <label for="input_mensagem">Mensagem</label><br/>
  <textarea name="mensagem" id="input_mensagem">
</textarea>
</form>
```

E-Mail

Nome

Mensagem

3. 10 Campos com suas etiquetas

3.5. Diferenças entre GET e POST

No formulário podemos usar tanto o método GET quanto o POST. Ambos vão enviar as mesmas informações ao servidor, que as entenderá de uma mesma maneira.

Como o GET colocar todos os dados coletados diretamente na URL, não é recomendado utilizar o GET quando essas informações são muito numerosas ou sensíveis (ex: senha).

O POST por sua vez, codifica a informação e coloca ela diretamente no corpo da requisição. Isso deixa um pouco mais segura a transmissão, além de não haver limite do tamanho da informação a ser transmitida.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O elemento form é o responsável por coletar e enviar as informações do usuário.
- Existem diversos campos para coletar as mais diferentes informações do usuário.
- O uso do método GET se torna inadequado para a coleta de informações, sendo recomendado o uso do POST.

Capítulo 4

Introdução ao CSS3

- Introdução
- CSS;
- Sintaxe do CSS
- Propriedades mais comuns.

4.1. Introdução

No começo da web, era importante apenas transmitir as informações, não importando se elas estavam 'bonitas'. Mas com a popularidade da web, surgiu a necessidade de formatar melhor, colocar cor, nos documentos da web.

O CSS veio com o intuito de deixar simples a formatação mesmo dos mais complicados documentos.

4.2. CSS

CSS significa Cascading Style Sheets (folha de estilo em cascata). Estilo define como os elementos HTML são exibidos. Estilos são normalmente armazenados em Folhas de Estilo (arquivo CSS). Estilos foram adicionados para resolverem um problema no HTML 4.0. Folhas de estilo externo pode economizar tempo de trabalho. Múltiplos estilos podem ser definidos em cascata dentro de um único documento.

Como as folhas de estilos podem ser especificadas:

- dentro de um elemento HTML (Inline)
- dentro do elemento <head> de uma página HTML
- em um arquivo CSS externo

4.2.1. CSS Inline

Usado para aplicar uma regra apenas para uma tag de uma página específica.

```
<body style="background: yellow;">  
  <h1> Olá Mundo! </h1>  
</body>
```

4.1 Style Inline

4.2.2. CSS Interno

Usado quando em um único documento há apenas um estilo. Definido na seção <head> </head> ou usando a tag <style> </style>.

```
<head>
  <style type="text/css">
    body {background: yellow}
  </style>
</head>
```

4. 2 Estilos interno de página

4.2.3. CSS Externo

Ideal quando aplicado em várias páginas. Muda todo o visual se uma página mudando apenas um arquivo. Cada página deve ter um vínculo com a folha de estilo.

```
<head>
  <link rel="stylesheet"
    type="text/css" href="meuestilo.css" />
</head>
```

4. 3 Folha de Estilo Externo

Repare nos atributos do marcador <link>.

- rel: especifica o relacionamento entre o documento HTML e o documento ligado. Para folhas de estilo, o valor do atributo é "stylesheet"
- type: especifica o tipo do documento. Para CSS, o tipo é texto (text/css)
- href: especifica a localização do arquivo CSS. Pode ser um recurso local ou um recurso externo.

No marcador link apenas o atributo rel é obrigatório. Mas para inserir uma folha de estilo, precisamos colocar a referência (href).

4.3. Sintaxe do CSS

A sintaxe CSS é definida em 3 partes: um seletor, uma propriedade um valor:

```
seletor {
  propriedade: valor;
}
```


Um seletor simples pode ser um marcador HTML, uma classe ou um id Exemplo utilizando um marcador HTML:

```
body {  
    background: yellow;  
}
```

Se o valor é composto de múltiplas palavras, deve ser colocado aspas entre o valor. Se desejar especificar mais de uma propriedade, estas devem ser separadas por ponto e vírgula. Para tornar seu estilo mais legível, descreva cada propriedade em cada linha.

4.3.1. Classes de Estilo

Classes de estilo permitem definir diferentes estilos para o mesmo tipo de elemento HTML. No arquivo CSS, o nome de uma classe é identificada com um ponto (.). Ex.: Dois tipos de parágrafos em seu documento.

```
<p class="right">  
    Alinhado à direita  
</p>  
<p class="center">  
    Centralizado  
</p>
```

```
p.right {  
    text-align: right;  
}  
p.center {  
    text-align: center  
}
```

Alinhado à direita

Centralizado

4. 4 Uso de classes

Você pode omitir o nome do elemento no seletor (.center {text-align: center}). Neste caso todo marcador com a classe center terá o estilo aplicado.

```
<h6 class="center">
  Cabeçalho
  centralizado
</h6>
<p class="center">
  Parágrafo também
  centralizado.
</p>
```

```
.center {
  text-align: center
}
```

Cabeçalho centralizado

Parágrafo também centralizado.

4.5 Uso de classe sem tag

O nome da classe e o valor do atributo class não podem iniciar com números. Não utilizar espaços para um nome de classe. Um marcador HTML pode receber mais de uma classe. Para isso, separa-se cada uma das classes por espaços, no atributo class.

4.3.2. Seleção por ID

É possível definir estilo para elementos HTML usando o seletor id. Ele é definido com um #.

```
<p id="para1">
  Vermelho
</p>
<p id="para2">
  Azul
</p>
```

```
p#para1 {
  color: red
}

p#para2 {
  color: blue
}
```

Vermelho

Azul

4.6 Exemplo de seleção por IDV

4.3.3. Agrupamento de seletores

Você pode agrupar seletores. Para agrupar seletores basta separá-los por vírgulas. O estilo dentro das chaves é aplicado a todos os seletores declarados.

4.3.4. Pseudo-Seletores

pseudo-classes e pseudo-elementos são usados para adicionar efeitos especiais em alguns seletores.

<pre> Link não visitado
 Link visitado
 Mouse sobre o link
 Link selecionado </pre>	<pre>/* link não visitado */ a:link { color: #FF0000 } /* link visitado*/ a:visited { color: #00FF00 } /* mouse sobre o link */ a:hover { color: #FF00FF } /* link selecionado */ a:active { color: #0000FF }</pre>	<p><u>Link não visitado</u></p> <p><u>Link visitado</u></p> <p><u>Mouse sobre o link</u></p> <p><u>Link selecionado</u></p>
--	---	---

4. 7 Exemplo de Pseudo-Seletor com as âncoras

4.3.4. Pseudo-Seletores

Vamos ver algumas das propriedades mais comuns.

4.3.4. Pseudo-Seletores

Existem 16,777,216 cores disponíveis nas CSS. As cores podem ser um nome pré-definido (red, blue, black). Podem ser definidas por valores RGB (red é o mesmo que rgb(255,0,0), o mesmo que rgb(100%,0%,0%)). Ou por um código hexadecimal (red é o mesmo que #ff0000 e o mesmo que #f00).

As cores podem ser aplicadas usando as propriedades color e background-color.

<pre><h1> Título </h1></pre>	<pre>h1 { color: #ffc; background-color: #009; }</pre>
--	--

Título

4. 8 Exemplo de cor de letra e de fundo

4.4.2. Tamanhos e unidades

Em CSS existem várias formas de especificar tamanhos:

- px é a unidade para pixel (ponto na tela).
- em é um elemento com tamanho x vezes da fonte atual do elemento pai. Se a fonte tem tamanho 12px, 2em tem 24px
- rem funciona da mesma maneira que a anterior, mas a sua referência é o elemento raiz do documento.
- % é a unidade para porcentagem.

Se nenhuma unidade for informada, o padrão é px

	CSS ▾	
<code><p> Texto </p></code>	<code>p.porc {</code> <code> font-size:70%</code> <code>}</code>	Texto
<code><p class="porc"> Texto </p></code>		Texto
<code><p class="px"> Texto </p></code>	<code>p.px {</code> <code> font-size:18px;</code> <code>}</code>	Texto
<code><p class="pt"> Texto </p></code>	<code>p.pt {</code> <code> font-size:10pt;</code> <code>}</code>	Texto
<code><p class="em"> Texto </p></code>	<code>p.em {</code> <code> font-size:2em;</code> <code>}</code>	Texto

4. 9 Exemplo do uso das unidades

4.4.3. Estilos de texto.

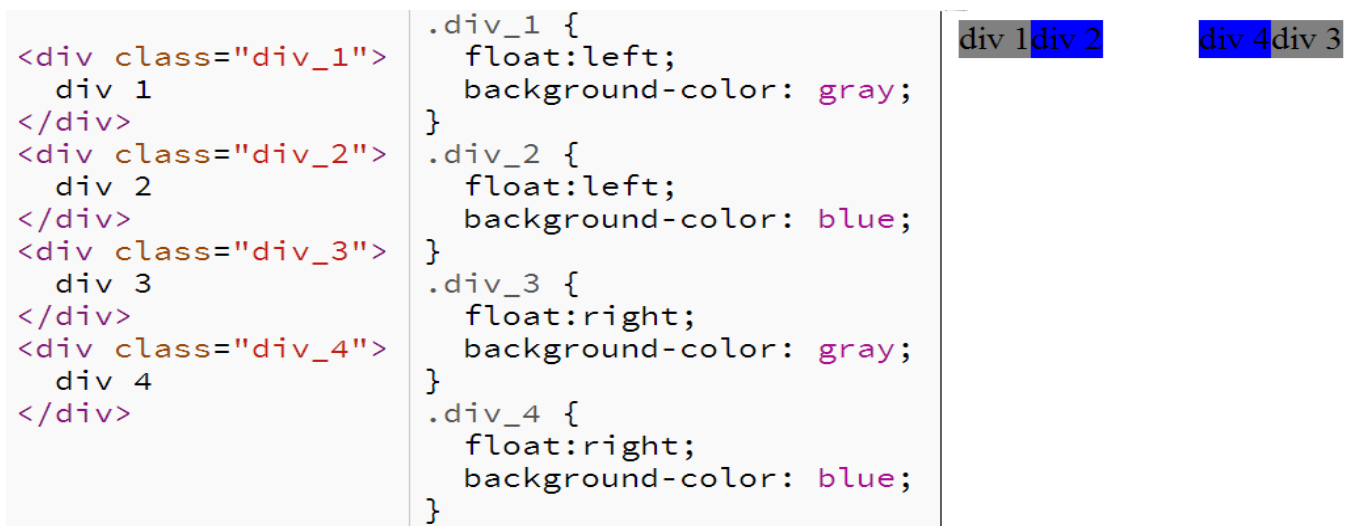
Os atributos que permitem mudar a forma como o texto é representado são: font-family, font-size, font-weight, font-style, text-decoration e text-transform, font-family indica a fonte a ser usada.

Esta fonte tem de estar no computador da pessoa que está a ver a página por isso não vale a pena usar fontes obscuras. Podemos especificar várias fontes separadas por vírgulas. O browser usa a primeira que o utilizador tenha. Se o nome da fonte tiver mais de uma palavra é necessário usar aspas. O último valor desta propriedade deverá ser uma classe de fonte mais genérica como serif, sans serif, cursive, fantasy, monospace.

4.4.4. Espacialização

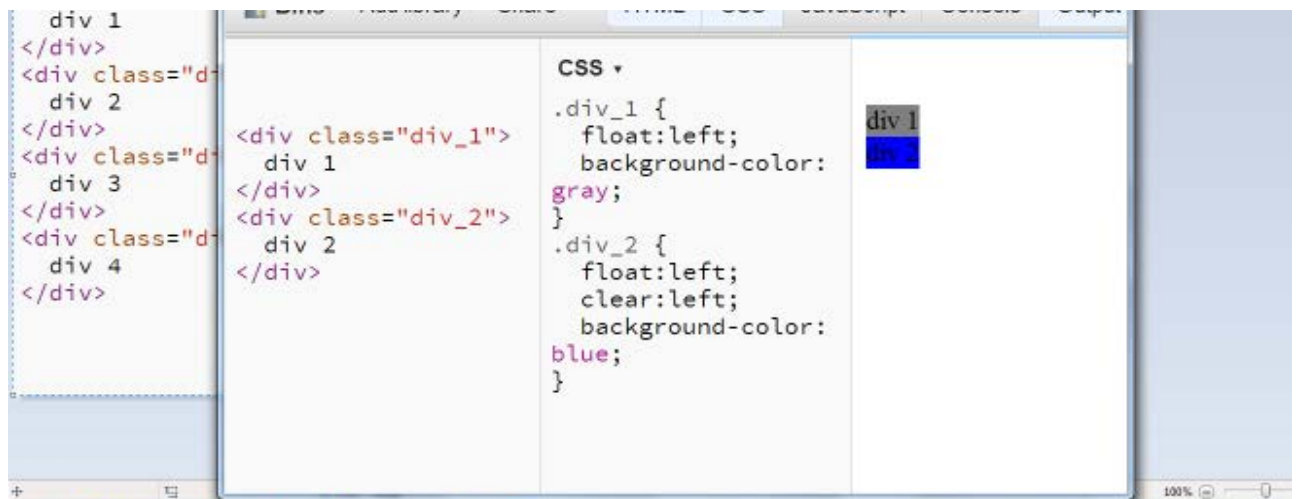
Para definir uma maneira de especializar um elemento em relação aos outros, temos a propriedade float e clear. O float define onde um elemento aparecerá em outro elemento. Pode assumir os valores left ou right.

O elemento com float irá mover-se mais à esquerda ou mais à direita possível



4. 10 Exemplo de uso do Float

Já o clear define em quais lados de um elemento outros elementos não podem flutuar (float). Assume valores none, left, right e both.



4. 10 Exemplo de uso do Float

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Apesar das três maneiras, a única recomendada é a utilização de arquivos CSS externos ao documento HTML.
- Seletores CSS são basicamente de três tipos: por tags, por classe de tags e por id de tags (além da combinação destes);
- Existem diversas propriedades do CSS, apenas a experiência e utilização lhe fará entender todas elas. Treine muito.

Capítulo 5

Layouts com CSS3

- Introdução;
- Trabalhando com dimensões;
- Display;
- Media Query;

5.1. Introdução

Com a disseminação de todo o tipo e tamanho de tela que são usados para navegar na web, foi-se necessário estender o CSS para que ele seja capaz de se adequar a tantas telas diferentes.

Técnicas mais modernas surgiram: flexbox, media query e uso de medidas relativas, com o objetivo de tornar essa tarefa mais fácil. Vamos aprender a todas.

5.2. Trabalhando com dimensões

Estas propriedades definem o tamanho dos elementos renderizados. Elas são height e width. Recebe valores em tamanho e porcentagem (mesmas unidades vistas anteriormente).



4. 10 Exemplo de uso do Float

5.2.1. Modelo Caixa (Box-Model)

Além da altura (height) e largura (width), temos algumas outras medidas que influenciam no tamanho total que um elemento HTML ocupa na página:

- Border: Já mostramos a borda, ela possui uma espessura, o que altera também o espaço ocupado pela tag HTML.
- Padding: Espaço entre o conteúdo da tag e a borda dela. Pode ser colocado para as quatro direções ou apenas as que forem interessantes (top, bottom, left, right).
- Margin: Espaço entre a borda do elemento até a borda do próximo elemento (ou de algum limite do documento). Também pode ser utilizada em todas as quatro direções ou apenas nas que forem interessantes.


```

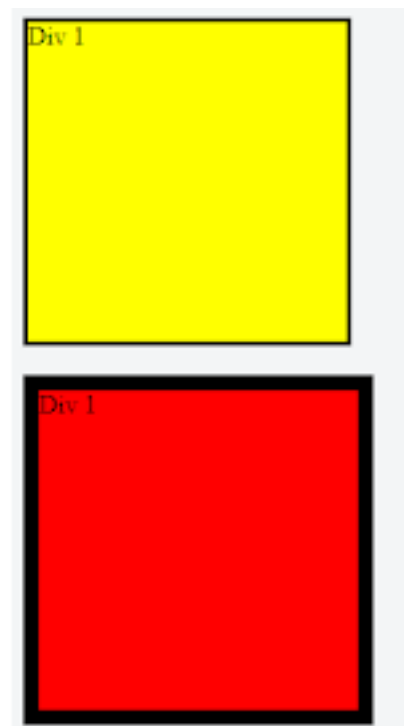
<div class="div1">
  Div 1
</div>
<br/>
<div class="div2">
  Div 1
</div>

```

```

1 div {
2   border: 3px solid black;
3   width: 200px;
4   height: 200px;
5 }
6 div.div1 {
7   background: yellow;
8 }
9 div.div2 {
10  background: red;
11  border-width: 10px;
12 }

```



5.2 Manipulação do Border

```

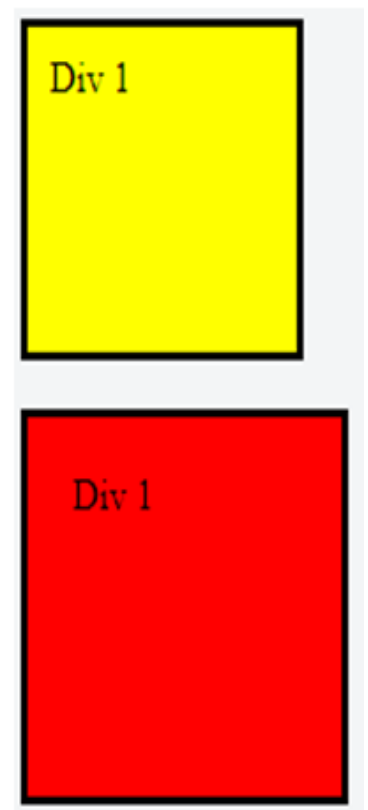
<div class="div1">
  Div 1
</div>
<br/>
<div class="div2">
  Div 1
</div>

```

```

1 div {
2   border: 3px solid black;
3   width: 100px;
4   height: 100px;
5 }
6
7 div.div1 {
8   background: yellow;
9   padding: 10px;
10 }
11
12 div.div2 {
13   background: red;
14   padding: 20px;
15 }

```

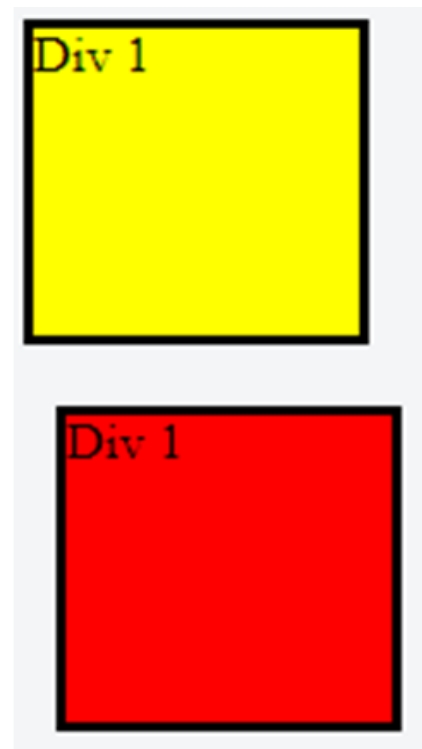


5.3 Manipulação do Padding

```
<div class="div1">
  Div 1
</div>
<div class="div2">
  Div 1
</div>
```

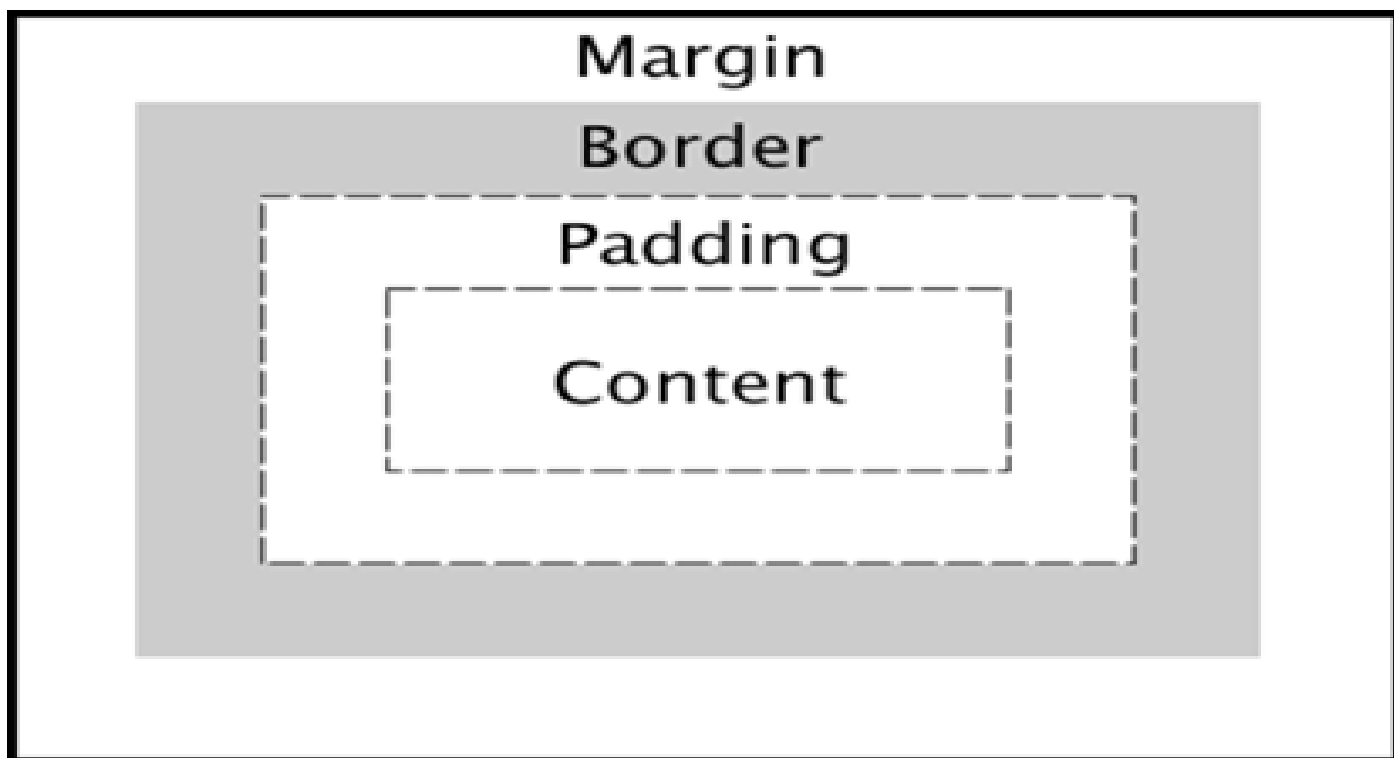
HTML ⚙

```
1 div {
2   border: 3px solid black;
3   width: 100px;
4   height: 100px;
5 }
6
7 div.div1 {
8   background: yellow;
9   margin: 10px;
10 }
11
12 div.div2 {
13   background: red;
14   margin: 20px;
```



5.4 Manipulação do Margin

Essas distâncias configuram o que chamamos de Box-Model (modelo caixa). Desta maneira, quando definimos os atributos height e width do elemento em si, esses valores não levam em consideração o padding e o border, fazendo com que os elementos sejam maiores do que deveriam.



5.5 Modelo Caixa

Para corrigir o problema de espaçamento e tamanho, basta utilizar a propriedade box-sizing. Os valores possíveis são

- content-box: Indica que as medidas de altura e largura não devem incluir padding e border.
- border-box: Indica que as medidas de altura e largura

Uma atitude comum entre os desenvolvedores é utilizar o border-box para todos os objetos como padrão. Para facilitar, podemos utilizar a seguinte regra para colocar esse atributo para todos os elementos da página usando o wildcard:

```
* { box-sizing: border-box }
```

5.3. Display

Define a maneira como os diversos elementos de uma página são exibidos. Temos quatro valores básicos:

- block: define uma região em bloco, que possui altura definida pelo conteúdo (ou CSS) e se expande da esquerda para a direita até onde for possível (ex: div's)
- inline: define uma região em linha, com a largura definida pelo conteúdo e altura pelo tamanho da linha (ex: span)
- inline-block: define uma região com tamanho controlada pelo conteúdo ou CSS, nenhuma tag funciona assim por padrão.
- none: define uma região invisível, ela existe no HTML mas não aparece para o usuário e não altera o layout.

```
<div class="div1">
  Div 1
</div>
<div class="div2">
  Div 1
</div>
<div class="div3">
  Div 3
</div>
<div class="div4">
  Div 4
</div>
```

```
div {
  border: 3px solid black;
  width: 100px;
  height: 100px;
}
div.div1 { background: yellow; display: block;}
div.div2 { background: red; display: inline; }
div.div3 { background: green; display: inline-block;}
div.div4 { background: black; display: none; }
```



5.6 Exemplos dos Displays

5.4. Media Query

O conceito principal de uma Media Query é verificar algumas informações do dispositivo visualizando o site, assim podendo definir um conjunto de regras que funcione melhor para este dispositivo. É possível verificar principalmente:

- Largura e altura do viewport (área visível da página).
- Largura e altura do aparelho em si (tela completa).
- Orientação (retrato ou paisagem).
- Resolução

Técnica de MediaQuery é muito popular no desenvolvimento mobile.

```
@media not|only tipomedia and (mediafeature) {  
    Regras CSS;  
}
```

5.7 Sintaxe de uma Media Query

5.5. Flexbox

No CSS3 foi definida mais uma propriedade display, a Flex(box). A ideia do Flexbox é garantir que o conteúdo interno da página consiga se reorganizar de uma maneira previsível caso a janela mude de tamanho. Ajuda a fazer layouts de maneira mais simplificada e fluída.

Propriedades que fazem parte do modelo flexbox são divididas em propriedades para o container (que possui o display flex, chamado de flex-container) e para os seus filhos (chamados de flex-items)

Para auxiliar em deixar previsível o comportamento dos elementos de um flexbox-container (em vermelho), temos uma série de propriedades envolvidas:

- flex-direction: direção que os itens devem seguir (coluna-linha).
- justify-content: como os itens se dispersam horizontalmente*.
- align-items: como os itens se dispersam verticalmente*.
- flex-wrap: se os itens devem passar a próxima linha caso falte espaço.
- align-content: como as linhas se dispersam.

Para os flex-items (em amarelo), precisamos das seguintes propriedades:

- **order**: identifica a ordem que o item deve aparecer.
- **align-self**: sobrecarrega o conteúdo da propriedade **align-items**.
- **flex**: identifica o tamanho deste elemento, em relação aos demais.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Apesar das três maneiras, a única recomendada é a utilização de arquivos CSS externos ao documento HTML.
- Seletores CSS são basicamente de três tipos: por tags, por classe de tags e por id de tags (além da combinação destes);
- Existem diversas propriedades do CSS, apenas a experiência e utilização lhe fará entender todas elas. Treine muito.

Capítulo 6

Manipulando Páginas com Java Script

- Introdução;
- Estrutura do JS;
- Sintaxe;
- Manipulação do DOM;
- Eventos.

6.1. Objetivo

Além de formatar e deixar com um bom layout, agora queremos dar vida aos documentos. Ações, efeitos, eventos, tudo que o JavaScript pode fazer por uma página HTML

Vamos ver como manipular eventos e como criar um novo conteúdo apenas com o javascript.

6.2. Estrutura do JS

O arquivo JS é um arquivo de texto com extensão .js. Neste arquivo são declarados:

- Variáveis: definem valores e armazenam dados
- Funções: definem comportamentos e ações para a página web
- Eventos: funções específicas disparadas a partir da interação do usuário com a página

O arquivo JS é basicamente uma sequência de comandos JS. Cada comando é executado pelo navegador na sequência. O código JavaScript pode ser inserido em uma página utilizando a tag script de duas maneiras, com o código diretamente dentro dela (não recomendado) ou usando um arquivo externo (recomendado, ex: `<script type="text/javascript" src="nome_arquivo.js"></script>`).

Qualquer código JS (dentro da página ou um arquivo externo) deve ficar entre os marcadores `<script>` e `</script>`. Um marcador HTML que DEVE ter a abertura e o fechamento (mesmo se for uma referência a um arquivo externo). Normalmente `<script></script>` é inserido no cabeçalho da página (`<head>`), mas também pode ser colocado no corpo (`<body>`). Atributos:

- type: informa que o script é um JS (não obrigatório no HTML5) (padrão: text/javascript)
- src: informa a localização do arquivo JS
- async: ativa a execução assíncrona
- charset: estabelece a codificação usada
- defer: se o script executa apenas quando a página acabar de carregar.

Uma página pode declarar quantos marcadores de script quiser. A leitura de uma página pelo browser é de cima para baixo. Isto inclui também a leitura do JS, como acontece com o CSS. Ou seja, tudo que for colocado em `/*script */` ou que estiver dentro de um arquivo .js será lido. Quando utilizamos um arquivo JS, o navegador faz uma requisição ao servidor (GET) para retornar o recurso

A cada chamada a uma página HTML o navegador também faz requisições para retornar os JS do servidor. Portanto, os arquivos ou códigos JS são incluídos como um anexo à sua página HTML. Assim como HTML e CSS, JS é um código executado no lado cliente.

6.3. Sintaxe

6.3.1. Variáveis

As variáveis em JS são “containers” para armazenar informação. Funcionam como a maioria das linguagens de programação.

- Operadores aritméticos: +, -, *, /, %
- Operadores lógicos: &&, ||, !
- Operadores de comparação: ==, !=, <=, <, >=, >
- Atribuição: =

Declarar/criar variável: `var x;` (Em JS não há tipos para declarar a variável, apenas `var`)

Quando criada, uma variável recebe valor `undefined`. Para atribuir um valor a variável, basta utilizar o operador de atribuição (`var x = 10`). Strings em JS podem ser atribuídas com aspas simples ou aspas duplas (mas usem aspas duplas!). É possível concatenar strings, números e variáveis em JS, utilizando “+”

6.3.2. Vetores

JS também suporta o uso de arrays. Para criar uma variável de array, utilizamos (`var alunos = new Array();`). Depois, basta inserir os valores nas posições do array.

- `alunos[0] = “Aluno1”;`
- `alunos[1] = “Aluno2”;`

Formas alternativas:

- `var alunos = new Array(“Aluno1”, “Aluno2”);`
- `var alunos = new [“Aluno1”, “Aluno2”];`

Repare que não é preciso declarar o tamanho do array.

6.3.3. Funções

JS permite o uso de funções, para deixar o código mais organizado. É um bloco de código executado quando é chamado, como acontece na maioria das linguagens de programação. Por exemplo, executar uma função quando clicar em um botão

Sintaxe básica:

```
function funcao() {  
    alert("Olá mundo!");  
}
```

Funções em JS podem receber argumentos, separados por vírgula

```
function funcao(arg1, arg2) {  
    alert(arg1);  
    alert(arg2);  
}
```

Funções em JS não declaram tipo de retorno. Entretanto, podem retornar algum valor para quem chamou. Utiliza-se a palavra chave return. Para executar a função, basta usar o nome e os parênteses: funcao();

As variáveis em JS tem dois escopos:

- Locais: declaradas em uma função
- Globais: Declaradas fora das funções

Funções são objetos no JavaScript e podem ser guardadas em variáveis.

6.3.4. Estrutura de Decisão

JS também define estruturas de controle (if...else if...else). O uso é quase idêntico à maioria das linguagens de programação:

```
if (condition1) {  
    //codigo  
} else if (condition2) {  
    //codigo  
} else {  
    //codigo  
}
```

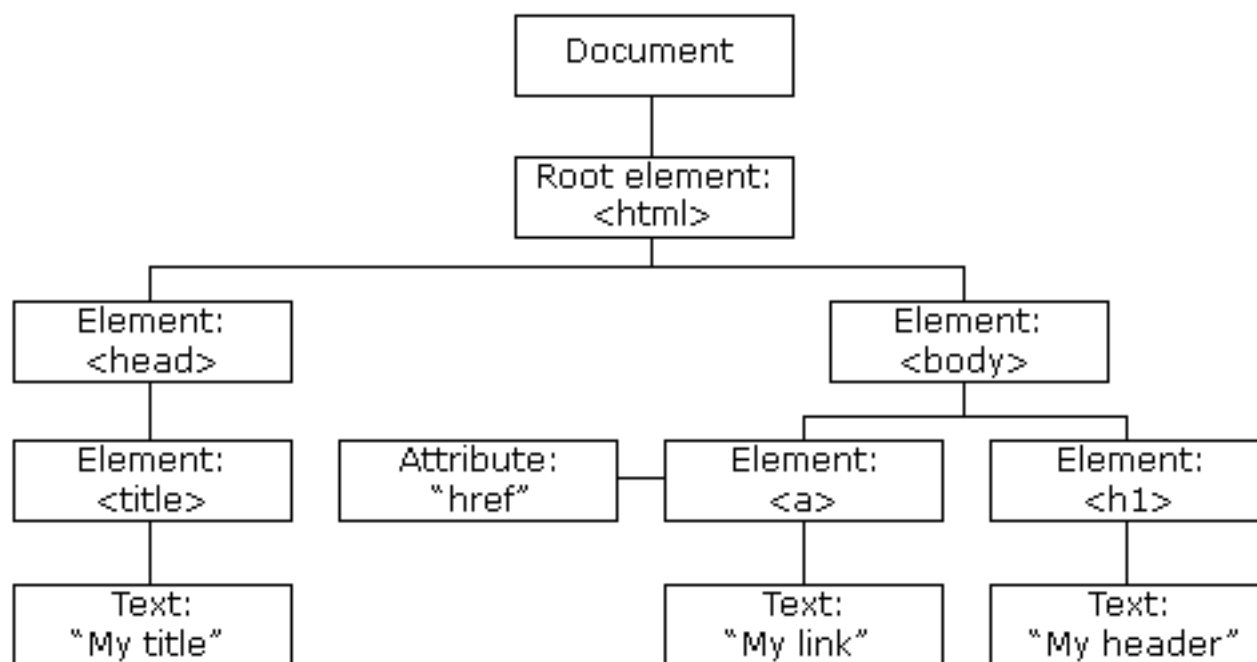
6.3.5. Estrutura de Repetição

JS também define estruturas de repetição, muito parecido com outras linguagens de programação

```
for (var i=0; i<10; i++) {  
    alert(i);  
}  
  
while (i<5) {  
    alert(i)  
}
```

6.4. Manipulação do DOM

Quando uma página é carregada o navegador cria um modelo de objetos que representa a página (DOM). O DOM é construído como uma árvore de objetos. Lembre-se que o HTML é basicamente um conjunto de marcadores aninhados. Um marcador pode ter marcadores, que pode ter marcadores.



6.1 Exemplo da estrutura do DOM

```
<!DOCTYPE html>
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <a href='#'>My link</a>
    <h1>My header</h1>
  </body>
</html>
```

6.2 Resultado do DOM

Porque estudar o DOM? JS consegue recuperar a estrutura da página em um objeto, ou seja, temos toda a estrutura da página em objetos dentro do JS. Assim, podemos manipular nossas páginas utilizando JS:

- Mudar o conteúdo de elementos HTML
- Mudar o estilo de elementos HTML
- Trabalhar com eventos
- Adicionar e remover elementos html

6.4.1. Como capturar um elemento HTML

Uma das grandes funcionalidades de JS é manipular a página HTML, para deixar as páginas mais dinâmicas. A primeira coisa que precisamos fazer é encontrar um elemento do HTML. Para isso, JS oferece um método chamado `document.getElementById()`. O `document.getElementById()` permite referenciar dinamicamente qualquer elemento do documento HTML através de um ID.

Existem outras formas de recuperar elementos HTML, não só pelo ID

6.4.1.1. Pelo nome do marcador

- `var parag = document.getElementsByTagName("p");`
- Repare que o argumento não é o valor do atributo name, mas sim o nome do marcador (p, div, span, h1, table)
- Portanto, forma este método retorna TODOS os elementos com o nome do argumento
- No exemplo, retorna todos os parágrafos da sua página
- Cada elemento é acessado na forma de um array
`parag[0]; parag[1];`

6.4.1.2. Pela classe

- `var classes = document.getElementsByClassName("classe");`
- Este método retorna TODOS os elementos com a classe enviada no argumento
- No exemplo, retorna todos elemento que tem o atributo `class="classe"`;
- Cada elemento é acessado na forma de um array `classes[0]; classes[1];`
- O array retornado possui o atributo `length` para informar quantos elementos foram encontrados

6.4.1.2. Por seletor CSS

- `var objetos = document.querySelectorAll("p.importantes");`
- Este método retorna TODOS os elementos que batam com o seletor escrito
- Cada elemento é acessado na forma de um array `objetos[0]; objetos[1];`
- O array retornado possui o atributo `length` para informar quantos elementos foram encontrados
- Se quiser retornar apenas o primeiro que bate com esse seletor, use a função:
- `var objeto = document.querySelector("p.importantes")`

6.4.2. Manipulações possíveis

A manipulação de objetos HTML do JavaScript pode:

- Escrever no documento HTML: `.write`
`document.write("<p>Parágrafo</p>");`
- Mudar conteúdo HTML de um elemento: `.innerHTML`
`document.getElementById("id").innerHTML="texto"`
- Mudar o valor de um atributo: `.nomeAtributo`
`document.getElementById("id").src="imagem.jpg"`

Modificar estilos CSS

- Cor de fundo: `.style.background`
`document.getElementById("id").style.background="#F0D";`
- Margin: `.style.margin`
`document.getElementById("id").style.margin="10px";`
- Margin-Top: `.style.marginTop`
`document.getElementById("id").style.marginTop="10px";`

6.5. Eventos

Eventos são ações que podem ser detectadas pelo JS. Todo elemento HTML possui eventos que podem ser disparados por funções JavaScript. Exemplos de eventos aplicados nos elementos:

- Clique o mouse – `onClick()`
- Passar o mouse - `onMouseOver()`
- Tirar o mouse– `onMouseOut()`
- E ainda muitos outros para combinar com formulários, como `onFocus()`, `onBlur()`, `onChange()`, `onSubmit()` e etc.

Esta forma de inserir o JS em um HTML constitui uma terceira abordagem de integração entre as linguagens. A diferença aqui é que só há disparo quando o evento acontece.

Para deixar a integração com separação de código, usa-se o recurso de função. Uma `function` agrupa comandos que são disparados quando o evento acontece. Também é possível enviar o elemento que recebe o evento como parâmetro da função

```
<script>
    function Alerta(){
        alert("Olá Mundo");
    }
</script>
<p onClick="Alerta()">Texto do parágrafo </p>
```

Também é possível definir os eventos direto no JS, e não nos marcadores.

```
<html>
<head></head>
<body>
  <input type='button' id="botao" value='Clique' />
  <script>
    document.getElementById("botao").onclick=function() {
      alert("clique!");
    };
  </script>
</body>
</html>
```

6. 4 Exemplo de evento diretamente no JS

6.6. Criando novo conteúdo no HTML

Também é possível inserir elementos no HTML utilizando os métodos:

- Cria e retorna um elemento HTML
var x = document.createElement("p");
- Adiciona o elemento criado em algum local no HTML
document.getElementById("id").appendChild(x)
- Coloca um texto dentro de um elemento
var t = document.createTextNode("Texto")

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Da mesma forma que o CSS, recomenda-se que os arquivos JS estejam separados do HTML.
- O JS é utilizado, sobretudo, para a manipulação do HTML;

Capítulo 7

Padrão MVC e o Django Framework

- Introdução;
- Padrões de Projeto
- Introdução ao Django Framework

7.1. Objetivo

Diversos padrões de projeto foram inventados para resolver os mais diversos tipos de problemas. Mas na web um se despontou ao longo dos anos, o MVC.

Vamos ver do que se trata esse padrão, como ele se encaixa no Django e introduzir a construção de aplicações com esse framework.

7.2. Padrões de Projeto

Padrões de projeto são compilados de boas práticas em desenvolvimento de software que atendem a um tipo específico de problema. São estratégias bem aceitas pela comunidade de desenvolvimento. De conhecimento global: diversos países, empresas e desenvolvedores usam os mesmos.

Possibilita um catálogo de soluções pré-existente para diversos problemas. Na web o padrão de projeto mais difundido e utilizado é o famoso MVC. Esse é um acrônimo para Model, View e Controller, ou em português, Modelo, Visualização e Controle.

7.2.1. MVC

A filosofia do padrão MVC é de trazer importantes características para projetos grandes:

- Separação de Responsabilidades: cada camada do sistema cuida das suas responsabilidades apenas (conexão com o banco, visualização de dados, controle de requisição);
- Baixo acoplamento: podemos mexer em cada camada sem que isso impacte diretamente na outra;

O MVC é o mais utilizado no mundo inteiro na web, mas não é o único:

- MVP: Model View Presenter.
- HMVC: Hierarchical MVC.
- MVVM: Model View ViewModel.

7.2.1.1. Modelo – Model

A camada de Modelo (Model) representa o negócio em que o software está inserido. Aqui existirão regras de negócio (ex: matrícula de aluno), representação de modelos/entidades (ex: aluno, professor), a conexão com o banco de dados, etc.

Basicamente qualquer coisa que envolva o negócio da aplicação deve estar nessa camada.

7.2.1.2. Visualização – View

A camada View/Visualização trata de como os dados e ações são mostrados aos usuários. Regras de interface gráfica, cores, espaçamentos, etc, estarão nessa camada.

Para a web, essa camada em geral é representada pelo HTML/CSS/JavaScript

7.2.1.3. Controle – Controller

O Controller/Controle atua como um meio de campo entre a View e o Model.

Ele é responsável por interpretar uma requisição HTTP, retirando os dados dela, enviar o que for necessário para o Model, e devolver uma resposta adequada à View.

7.3. Introdução ao Django Framework

O Django é um framework de desenvolvimento web feito em Python. Framework: Conjunto de ferramentas e técnicas para desenvolver algo de maneira rápida e fácil.

Foi desenvolvido seguindo o conceito da “não reinvenção da roda”, ou seja, o objetivo é usar as melhores práticas e componentes já desenvolvidos para acelerar ao máximo a produtividade do desenvolvedor. A versão 2 (mais atual) funciona apenas com o Python 3, a versão 1.11 ainda suporta o Python 2

Podemos listar as seguintes vantagens da utilização do Django:

- Desenvolvimento inicial rápido (tutorial pode ser feito em pouco mais de 20 minutos)
- “Baterias Inclusas” : As tarefas comuns já estão prontas para vocês: autenticação, administração de conteúdo, RSS, mapeamento do BD, etc.
- Segurança: Já possui defesas contra os tipos comuns de ataques (SQL Injection, CSRF, Clickjacking, Cross-Site Scripting)
- Escalável: aplicação decomposta em vários componentes, facilitando a escalabilidade, além de um utilitário nativo de cache

7.3.1. MVC no Django (MTV)

O Django está dividido nas seguintes camadas:

- Camada de Modelo (Model Layer): Estarão presentes os objetos representando as abstrações dos modelos do sistema desenvolvido. Cuida da conexão com o banco de dados, bem como das execuções das query's no banco. Igual ao Model do MVC.
- Camada de Visualização (View Layer): Recebe as requisições do cliente e processa as informações necessárias, devolvendo para o cliente uma resposta com algum conteúdo HTML ou um código de erro HTTP. Igual ao Controller do MVC.
- Camada de Templates (Template Layer): Uma sintaxe para produzir o seu conteúdo HTML de uma maneira dinâmica e fácil. Igual ao View do MVC.

7.3.2. Instalação e preparação.

Para iniciar um projeto novo com o Django usamos a ferramenta django-admin. Esse comando é uma interface com o módulo de administração do Django (tem uma interface web também, veremos mais para frente). Para iniciar então, fazemos o comando: `django-admin startproject ${NOME_PROJETO} ${DIRETORIO}`

Quando o comando for rodado, teremos a seguinte estrutura de projeto:

```
projeto\                                # Raiz do projeto
  venv\                                  # Pasta do ambiente virtual
  projeto\                              # Pasta do projeto
    settings.py                         # Arquivo de configurações gerais
    urls.py                             # Arquivo de configuração de URL's
    wsgi.py                             # Arquivo de configuração do WSGI/Apache
    __init__.py                         # Arquivo de modularização
  requirements.txt                      # Arquivo de requisitos
  manage.py                             # Arquivo de gerenciamento de projeto
```

Para testarmos se o Django foi criado corretamente, podemos executar no `manage.py` o servidor local: `python manage.py runserver`.

7.3.3. Aplicações e Projeto

No Django, dividimos nossos projetos em aplicações. Aplicações são definidas como módulos que fazem alguma coisa (ex: login, compra, checkout). Portanto um projeto é um conjunto de configurações e de aplicações.

Aplicações são consideradas como módulos Python, portanto um projeto pode conter múltiplas aplicações, e uma aplicação pode aparecer em mais de um projeto.

Não existe um consenso em como separar os projetos nessas aplicações. Tente agrupar as funcionalidades do seu sistema/projeto. Possíveis separações para um e-commerce:

- Autenticação
- Busca de produtos
- Checkout
- Área do comprador
- Relatórios
- Carrinho de compras

É comum que os projetos tenham uma aplicação central ou principal (core), juntando todas as outras aplicações disponíveis. Para criar uma aplicação no Django, devemos rodar o comando `python manage.py startapp core`.

Isso cria uma pasta chamada core com vários arquivos dentro dela:

- `admin.py`: arquivo de administração da aplicação
- `apps.py`: configuração da aplicação para o administrador
- `migrations\`: pasta onde as modificações do banco de dados são guardadas
- `models.py`: configuração dos modelos a serem utilizados no projeto
- `tests.py`: arquivo base de testes da aplicação
- `views.py`: arquivo de visualizações do Django
- `__init__.py`: arquivo de módulo do Python

Para terminar de 'instalar' a nova aplicação, devemos dizer isso no arquivo `settings.py` do projeto. Procure a propriedade `INSTALLED_APPS` e adicione o nome da nova aplicação ('core')

7.3.4. Primeira view – Olá Mundo!

Vamos construir a primeira visualização do projeto, o index. O Django define uma visualização, ou view, como o conjunto de dados que serão apresentados para o usuário e não a maneira que eles serão apresentados, ou seja, uma view para o Django é uma função Python que recebe uma requisição HTTP e retorna os dados para o cliente. Views são funções escritas no arquivo `views.py` das suas aplicações.

Toda função/view deve receber como parâmetro um objeto com os dados da requisição enviada ao Django. Lembrando, views no Django são funções Python que recebem uma requisição do cliente e devolvem uma resposta com os dados necessários.

Na web, uma requisição é sempre abstraída em um objeto do tipo `HttpRequest` e a resposta em um objeto do tipo `HttpResponse`.

```
# views.py
from django.shortcuts import render
from django.http import HttpResponse

def index(request):
    return HttpResponse("Olá Mundo!")
```

7.1 View *olá mundo!*

Para que a primeira view funcione, basta entrar com um novo mapeamento de URL (`url(^$, core.view.index)`)

7.3.5. GET e POST no Django

Para a nossa view `index` é razoável que ela apenas trate o caso da requisição com o método GET, afinal ela apenas deve devolver uma página HTML.

E para um formulário? Se temos um formulário de login na página `http://localhost:8000/login`, como podemos diferenciar o GET (retorno do HTML) com o POST (submissão do formulário). A solução está no parâmetro `request` das views.

Cada view no Django recebe um parâmetro com os dados da requisição. Esses parâmetros são, em grande parte, aqueles que conseguimos ver usando o DeveloperWebTools do Chrome:

- `request.method`: devolve qual método HTTP (GET, POST, DELETE...)
- `request.path`: devolve o caminho dentro do seu domínio (/login, /ads)
- `request.GET`: devolve um `QueryDict` com os dados passados no GET
- `request.POST`: devolve um `QueryDict` com os dados passados no POST

Para acessar os parâmetros (seja no `request.GET` ou no `request.POST`) podemos usar o método `get`: `request.POST.get("nome-do-parametro")`

O nome do parâmetro que deve ser usado é aquele que está no atributo `name` dos inputs do formulário: `request.POST.get("email")`

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Mesmo depois de tanto tempo, o padrão dominante da WEB é o MVC.
- O Django utiliza um padrão próprio (MTV), muito similar ao padrão MVC
- O Django é um framework baseado em pequenas aplicações, para permitir uma melhor divisão de responsabilidades.

Capítulo 8

Estilo CapX

- Introdução;
- Sistema de templates;
- Template Tags
- Static Files;
- Herança de Templates

8.1 Introdução

É comum que não web nós tenhamos muitas páginas que sejam bem parecidas umas com as outras, que um pouco de lógica de programação permitiria que de uma única página base nós conseguíssemos montar várias, mudando apenas alguns parâmetros.

Para esse objetivo temos o sistema de templates do Django, misturando HTML e lógica Python para facilitar a produção de páginas.

8.2 Sistema de Templates

Para poder construir páginas dinamicamente, o Django possui uma engine de templates nativos (a partir da versão 1.8).

Esse sistema (engine) é composto de:

- Arquivos modelos html's (templates files),
- Tags específicas (template tags),
- Filtros para alterar o comportamento das tags (template filters)
- Um contexto de requisição (context).

O funcionamento segue da seguinte forma:

A requisição chega na sua view (por exemplo index).

- A view processa todas as informações (banco de dados) e obtém os dados necessários.
- Ela monta um objeto de contexto (context), que é basicamente um dicionário Python, com as informações processadas.
- A view renderiza o template escolhido (ex: "index.html") passando o contexto criado (objeto context).
- O renderizador procura as template tags dentro do arquivo e troca seus valores pelos presentes no objeto de contexto.
- A página é enviada ao cliente.

8.3 Template Tags

Para mostrar informações dinâmicas nas páginas do Django, usamos as templates tags juntamente com o contexto da página. Para imprimir variáveis com template tag utilizamos o marcador `{{ }}`.

Outras tags utilizam o marcados `{% %}`. Veremos adiante. Caso seja necessário apenas imprimir uma variável, basta usar a template tag com o nome dessa variável (ex: `{{ titulo }}` imprime a varável titulo).

Existem muitas template tags embutidas no Django, para diversas tarefas:

- Mostrar conteúdo: `for`, `context`, `cycle`
- Mudar exibição da página: `if`, `comment`, `autoescape`
- Incluir ou excluir conteúdo: `include`, `load`
- Definir regiões de conteúdo: `block`

Para ver todas as template tags definidas: <https://docs.djangoproject.com/en/1.11/ref/templates/builtins/#built-in-filter-reference>

Veremos primeiro duas das mais comuns: `for` e `if`. Conforme avançarmos com os estudos veremos as outras.

8.3.1. Template Tag – ForLoop

Assim como no Python, podemos iterar sobre um resultado em lista para produzirmos uma sequência de resultados na tela. Para isso temos o template tag `ForLoop`.

Podemos adicionar na variável de contexto, um atributo cujo valor seja uma sequência (lista) e, com isso, usamos uma estrutura muito similar a do `for` do Python.

```
{% for item in sequence %}
  {{ item }}
{% endfor %}
```

8.0 Exemplo de ForLoop

Nessa tag, item é uma variável temporária contendo algum valor dentro da sequência sequencia, da mesma maneira que no Python.

No Django, toda tag com a estrutura {% ALGO %} deverá ter uma estrutura finalizando a sua execução com {% endALGO %}.

Como definimos uma variável item dentro do for, podemos imprimir ela usando {{ item }}, nesse caso filtros podem ser utilizados também.

8.3.2. Template Tag – If-Elif-Else

Da mesma forma que a tag ForLoop é bem similar ao for do Python, temos a tag If-Elif-Else no Django para simular a estrutura If-Elif-Else do Python.

Essa tag serve para apresentar algum conteúdo na página que dependa de alguma condição booleana. Sua estrutura é muito similar ao do Python.

```
{% if condicao1 %}  
    alguma coisa  
{% elif condicao2 %}  
    outra coisa  
{% else %}  
    outra coisa ainda  
{% endif %}
```

8.0 Template tag If Elif Else

Da mesma forma que no Python, a expressão condicao1 deve retornar um booleano para ser avaliado, caso seja True ele entra na estrutura, caso contrário passa para a próxima avaliação (Elif) e assim por diante.

Segue as mesmas regras de construção do If-Elif-Else do Python:

- Bloco if é obrigatório, sempre deve começar por ele.
- Bloco elif não é obrigatório, vem depois do bloco if e podem existir quantos forem necessários.
- Bloco else não é obrigatório, deve ser sempre o último bloco e só pode haver um.

8.4. Static Files

Ao definir a `STATIC_URL` podemos usar os arquivos estáticos em caminhos como `/static/css/home.css`.

Caso essa URL mude, teríamos que entrar em TODAS as páginas que usam conteúdo estático (ou seja todas) e alterar a URL.

Para facilitar essa manutenção e auxiliar ao fazer a mudança para produção, temos um conjunto de tags na biblioteca static para usar nas nossas páginas.

Para importar bibliotecas de templates tags adicionais, como as que estão em outras aplicações ou mesmo as customizadas, temos a tag `{% load NOME %}`. Para o caso dos arquivos estáticos, vamos usar a biblioteca: `{% load static %}`.

Com a biblioteca carregada, podemos usar as tags definidas internamente nelas. No caso, podemos trocar todas as urls do tipo

- `“static/css/home.css”`

Por

- `“{% static ‘css/home.css’ %}”`

8.5. Herança em Templates

Vamos criar um arquivo chamado `base.html`. Nele vamos definir as regiões comuns e não comuns das páginas html do sistema. No arquivo `base.html`, aquilo que for a parte comum (header e footer por exemplo), nós mantemos o conteúdo html normalmente.

Na parte não comum, por exemplo a parte do main, nós vamos substituir o conteúdo pela tag `block` para definir uma região: `{% block container %} {% endblock %}`

A tag `{% block NOME %}{% endblock %}` define uma região no seu template base que vai receber algum conteúdo de quem estendê-lo. O atributo `NOME` define um nome para identificar a região de qual conteúdo ela irá receber.

Se existe uma tag `{% block NOME %}{% endblock %}` no `base.html`, ela será substituída pelo conteúdo dentro da tag `{% block NOME %}{% endblock %}` do arquivo que estender o `base.html` (ambos os blocos com o mesmo nome!) Se não houver nenhum bloco com esse nome no arquivo estendido, o bloco ficará vazio.

Se houver conteúdo no bloco, por exemplo `{% block NOME %} Conteúdo {% endblock %}` esse conteúdo será usado como um conteúdo padrão caso o bloco não esteja presente no novo arquivo.

Podemo usar uma variável de referência dentro da tag `block`, variável chamada também de `block`, para acessar e utilizar esse conteúdo padrão, usando a variável `{{ block.super }}`.

O `block.super` carrega a informação padrão definida no `base.html`, portanto se o valor padrão definido é `Teste`, ao escrever no `base.html`: `Outro {{ block.super }}` Temos como resultado: `Outro Teste`.

Teoricamente, podemos ter quantos níveis de herança de templates quisermos. Por exemplo, temos o `base.html`, poderíamos ter um `base_produto.html` que estende o `base.html`. Poderíamos ter o seguinte layout:

- `base.html`
- `base_produtos.html`
- `base_contato.html`
- `base_produtos_tipo1.html`
- `base_produtos_tipo1_caso_1.html`
- `base_produtos_tipo1_caso_2.html`
- `base_produtos_tipo2.html`

Dessa forma, a quantidade de níveis pode crescer descontroladamente. Para evitar isso, existe o padrão definido de três camadas:

- Só existe um `base.html` que o mais geral possível dentro do projeto.
- Se houver necessidade de outro template genérico para o projeto, este deve estender o `base.html`.
- Se houver templates específicas nas aplicações, eles devem ter uma base específica dentro da aplicação, ex: `base_core.html`. Todos os templates da aplicação devem estender o `base_<app>` e somente ele.
- O terceiro e último nível são as páginas de fato (views)

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O sistema de templates mistura lógica Python (if ou for) com códigos HTML para facilitar a produção de páginas.
- O sistema também possui uma maneira de prover arquivos estáticos para a suas páginas.
- É possível herdar outros templates no Django.

Capítulo 9

Mapeamento de URL's

- Introdução;
- Sistemas de URL's
- Parâmetros por QueryParameter
- Parâmetros dentro da URL

9.1 Introdução

Na sua concepção, a URL servia apenas para indicar um caminho pela rede para chegar a esse recurso. Esse caminho era basicamente um caminho pelas pastas de um servidor de arquivos e o arquivo final.

Agora, cada vez mais os conteúdos buscados na internet são dinâmicos. As urls agora representam conteúdos que só são 'criados' quando são pedidos pelas urls, e agora elas são capazes de carregar mais informações do que um simples caminho.

9.2 Sistema de URL's

O Django possui um mapeamento obrigatório de URL's. Esse mapeamento diz qual URL (ou pedaço) é redirecionado para qual view construída. Para esse mapeamento é utilizada uma String específico, chamada de expressão regular.

9.2.1 Expressões Regulares

Expressões regulares são Strings construídas para procurar regiões em textos com características específicas. Por exemplo, gostaríamos de buscar todos os números que existam no seguinte texto: "Meu nome é João, tenho 33 anos e jogo futebol 2 vezes no mês"

O módulo do Python que cuida de regex é o re. No exemplo anterior, vamos separar números de palavras usando REGEX. Na String que define a regex, temos alguns símbolos:

- r – define que a String que segue é uma expressão regular.
- \d – define a busca por qualquer dígito (entre 0 e 9).
- + - define que a busca deve procurar repetições do conteúdo (ao menos 1).
- [] – define agrupamento de opções de busca (separadas por vírgula).
- A-Z – define a busca por qualquer letra maiúscula.
- a-z - define a busca por qualquer letra minúscula.

```
import re
frase = 'Meu nome é João, tenho 33 anos e jogo futebol 2 vezes no mês'
numeros = re.findall(r'\d+', frase)
palavras = re.findall(r'[A-Z,a-z]+', frase)
print(numeros) #Imprime [33,2]
print(palavras) #Imprime o resto
```

9.1 Exemplo de uso de expressões regulares

Vamos olhar uma URL do Django para entender como o regex se aplica: `url(r'^admin/', admin.site.urls)`. Lembrando que o `r` mostra que a String `'^admin/` é uma regex. Temos um novo caracter: `^` - mostra que o resultado deve começar com a String descrita.

Nesse caso, teríamos o seguinte resultado para essa regex:

- `/admin` – Passou!
- `/admin/legal` – Passou!
- `/admin/mais/legal` – Passou!
- `/dadmin/` - Não Passou!

Podemos então usar combinação de regex para obter as urls que quisermos. E para obter parâmetros da URL. Temos duas maneiras: a clássica e menos usada e a moderna e mais usada.

9.3 Parâmetros por QueryParameter

A maneira clássica é por QueryParameter das url's. Por exemplo, queremos mostrar os detalhes de um curso específico em uma página. Temos a seguinte configuração então:

- A URL de cursos é (urls.py): `url(r'^curso', curso)`
- Uma view que recebe o curso (views.py)

Agora queremos buscar o curso por sigla pela seguinte url:

<http://localhost:8000/curso/?sigla=ADS>. Para capturar esse parâmetro da URL, vamos usar a função `request.GET.get('sigla')`.

Apesar de funcional, essa maneira está deixando de ser usada. Entre outros motivos, o mais importante é o fato dos indexadores (Google) ignorar esses parâmetros na busca deles. Esses indexadores em geral ignoram quaisquer parâmetros que venham após a interrogação.

9.3 Parâmetros por QueryParameter

A maneira clássica é por QueryParameter das url's. Por exemplo, queremos mostrar os detalhes de um curso específico em uma página. Temos a seguinte configuração então:

- A URL de cursos é (urls.py): `url(r'^curso', curso)`
- Uma view que recebe o curso (views.py)

Agora queremos buscar o curso por sigla pela seguinte url:

<http://localhost:8000/curso/?sigla=ADS>. Para capturar esse parâmetro da URL, vamos usar a função `request.GET.get('sigla', '')`.

Apesar de funcional, essa maneira está deixando de ser usada. Entre outros motivos, o mais importante é o fato dos indexadores (Google) ignorar esses parâmetros na busca deles. Esses indexadores em geral ignoram quaisquer parâmetros que venham após a interrogação.

9.4 Parâmetros dentro da URL

A maneira mais moderna para isso é usar o parâmetro dentro da URL:

- <http://localhost:8000/curso/ads>
- <http://localhost:8000/curso/bd>

A esse tipo de parâmetros damos o nome de Path Parameter. Para fazer isso, precisamos alterar o mapeamento da URL para capturar uma região da url, usando a regex de captura (os parênteses), por exemplo, `url(r'^curso/([A-Z,a-z]+)', curso)`.

Os parênteses marcam uma região a ser capturada pelo Django. Essa região vai capturar qualquer letra escrita após a barra. Na view, cada parâmetro capturado na URL vai aparecer como parâmetro de entrada na função, juntamente da requisição.

```
...
def curso(request, sigla):
    contexto = {
        "curso": "Escolhido o curso com sigla " + sigla
    }
    return render(request, "curriculo/curso.html", contexto)
...
```

9. 2 Exemplo de chamada com parâmetro.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- A URL tem a possibilidade nos indicar mais do que apenas um domínio e página
- Os caminhos dentro de uma URL não são mais apenas pastas em um servidor, são divisões lógicas dentro de uma aplicação complexa.
- Os caminhos ainda podem trazer informações pertinentes de um conteúdo dinâmico de uma entidade de um sistema.

Capítulo 10

Conectando o BD no Django

- Introdução;
- Conectando no Banco de Dados;
- Mapeamento ORM;
- Migrações;

10.1 Introdução

Mesmo não sendo obrigatório, é comum que nossas aplicações na WEB se conectem a um ou mais bancos de dados. A proximidade do mundo relacional com o orientado a objetos torna praticável um mapeamento automático entre ambos.

Vamos ver como conectar o Django nos mais diversos SGBD e como usar o mapeamento ORM a nosso favor.

10.2. Conectando no banco de dados

Todo framework possui suas maneiras de conectar a alguns bancos de dados. Conectamos ao BD - Banco de Dados - para salvar (ou persistir) os nossos dados das aplicações.

Para conectar aos BD's, as linguagens de programação utilizam drivers de conexão. Ao conectar no BD, temos API's genéricas de acesso, leitura e escrita no BD.

No arquivo settings.py alteramos as configurações de conexão em banco de dados. O Django vem por padrão com o SQLite (v3). É um micro BD com persistência e consistência de dados, perfeito para testes e aplicações aninhadas (embedded).

O Django oferece suporte aos BD's:

- PostgreSQL (pip install psycopg2)
- SQLite v3 (nativo)
- MySQL (pip install mysqlclient)
- Oracle

Através de bibliotecas instaladas via pip, mantidas por outras empresas, temos suporte também a:

- Sybase ASE
- SQLServer
- MySQL (Python 3)
- Firebird
- IBM DB2

10.3. Mapeamento ORM

No mundo da orientação a objetos, é comum fazermos uma ponte, ou tradução, do modelo relacional para o modelo de objetos. Para auxiliar nessa transição, os frameworks costumam dispor de ferramentas auxiliares. Essas ferramentas são chamadas de ORM - Object Relational Mapping, ou Mapeamento Objeto-Relacional.

Essas ferramentas utilizam as melhores práticas tanto de modelagem relacional quanto orientação a objetos para criar um mapa entre os dois modelos. Em geral temos dois mapeamentos possíveis:

- **Mapeamento Tabelas - Classes:** A ferramenta escaneia todas as tabelas criadas e cria classes para todas as tabelas primárias (não relacionamento), os relacionamentos em chave estrangeira são representados por composição de objetos.
- **Mapeamento Classes - Tabelas:** Mapeia as suas classes criadas e constrói tabelas para cada uma. Composição de objetos são tratadas como relacionamentos em chave estrangeira.

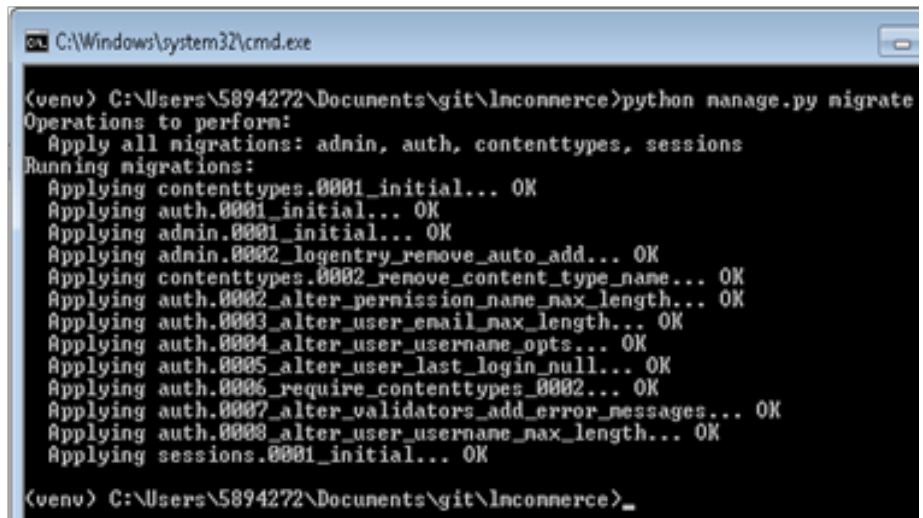
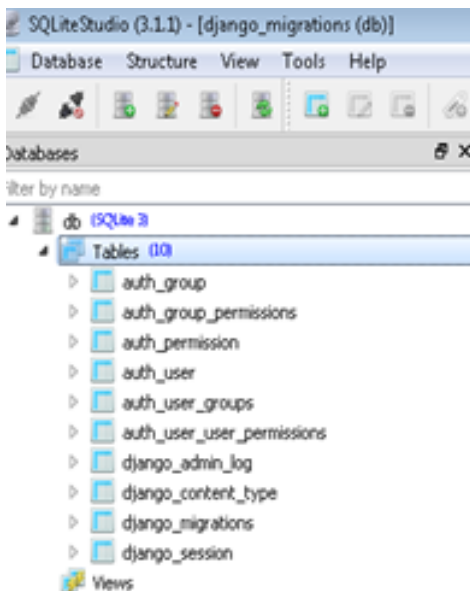
Características de uma ferramenta ORM:

- Mapeamento de duas mãos (classe - tabela).
- Gerenciamento automático de conexões com o BD.
- Identificação em classes que serão gerenciadas pela ORM (annotations ou metadados).
- Geração automática de SQL.
- Utilização de chave artificial numérica é preferível (em algumas é obrigatória).

A ORM do Django já vem instalada nativamente. Utiliza o conceito de migrations (migrações). Migrações são arquivos de diferenças entre dois estados do seu modelo (tabelas ou classes), ou seja, sempre que você alterar as suas classes é possível gerar um arquivo de diferenças no SQL (e vice-versa).

10.4. Migrações

Por enquanto não temos nenhuma classe nossa para migrar para o Banco de Dados. Mas e todas aquelas aplicações já instaladas no Django? Elas possuem classes e modelos próprios, portanto podemos migrá-las para o BD. Execute o seguinte comando `python manage.py migrate` e veja o resultado.



10.1 Migrações

Vemos que o Django já aplicou uma série de migrações vindas de várias aplicações padrões (auth, admin, sessions).

As classes próprias a serem migradas devem estar nos arquivos models.py das aplicações. O arquivo por enquanto está com apenas um import. Esse import é importante pois todo modelo que é gerenciado pelo Django deve herdar da classe models.Model

Dentro desse arquivo vamos definir os nossos modelos do Django. Como o Python é dinamicamente tipado, não teríamos como definir os tipos dos atributos para as tabelas. Para isso, o Django possui diversos Model Field Types (Tipos de campos de modelo). Esses Field Types fazem com que o modelo consiga traduzir um atributo para uma coluna do SQL. Existem quase 30 tipos definidos. Alguns dos tipos comuns de SQL:

- CharField: Referente ao VARCHAR do SQL
- DateField e DateTimeField: Referente ao DATE e DATETIME do SQL
- DecimalField: Referente ao DECIMAL do SQL
- IntegerField e BigIntegerField: Referente ao INT e BIGINT do SQL
- TextField: Referente ao TEXT do SQL

Um exemplo de primeiro modelo:

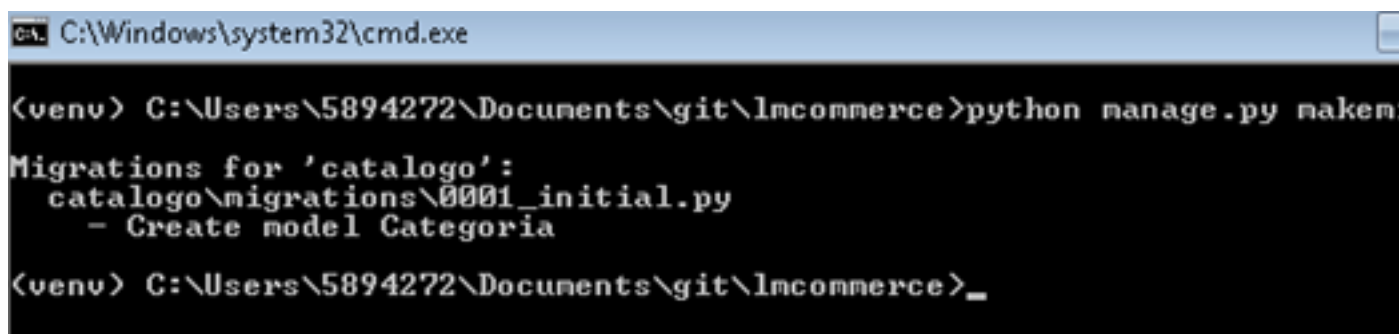
```
from django.db import models
class Categoria(models.Model):

    nome = models.CharField("Nome", max_length=50)
    etiqueta = models.SlugField("Etiqueta", max_length=50)
    def __str__(self):
        return self.nome
```

10.2 Primeiro modelo

No nosso primeiro modelo criamos uma classe com nome e etiqueta e a função de representação (`__str__`). Todo `FieldType` possui como primeiro parâmetro o `verbose_name`, que seria um texto descritivo curto sobre o atributo. Depois vem uma sequência de parâmetros nomeados para configurar o campo (ex: `max_length`).

Ao criar um primeiro modelo, vamos criar a sua migração para o BD. Rode o comando `python manage.py makemigrations` Dentro das pastas migrations das aplicações que contiverem modelos novos ou alterados, vão aparecer alguns arquivos gerados.



```
C:\Windows\system32\cmd.exe

(venv) C:\Users\5894272\Documents\git\lmcommerce>python manage.py makemigrations
Migrations for 'catalogo':
  catalogo\migrations\0001_initial.py
    - Create model Categoria

(venv) C:\Users\5894272\Documents\git\lmcommerce>_
```

10.3 Migração de novo modelo

Esses arquivos vão conter as diferenças encontradas nos modelos e no BD, toda vez que alteramos o modelo de alguma maneira e quisermos replicar isso no BD, devemos rodar o comando `python manage.py makemigrations` para ver se tudo está certo.

Caso a classe contenha algum erro de interpretação, durante o comando vamos saber quais erros existem. Se tudo estiver certo, podemos rodar o `python manage.py migrate` e olhas novamente o SQLite Studio.

10.5. Manipulação de dados via ORM

Para testar se o acesso está correto e eficiente, temos um comando presente: `python manage.py shell`. Isso abre um shell do Python com as informações e configurações do Django já carregadas. A primeira coisa a fazer é importar os nossos modelos (`from core.models import *`)

Para cada modelo que estende o objeto `models.Model`, o Python adiciona uma série de ferramentas embaixo do objeto `objects`. Ao mandar imprimir o `objects` do modelo, temos a resposta. Esse objeto `Manager` vai ser nossa interface com o BD

Vamos analisar os métodos mais gerais do `Manager`:

- `all()`: Lista todas as entradas desse tipo.
- `create(...)`: Cria uma nova entrada desse tipo.
- `get(...)`: Obtém uma única entrada baseada em um critério (deve voltar um).
- `filter(...)`: Lista todas as entrada que obedecem algum critério

10.6. Relacionamentos entre objetos

Em via de regra temos apenas três tipos: muitos para um, muitos para muitos e um para um. Assim como na modelagem de banco de dados, o ORM mapeia essa relação entre objetos colocando uma referência de um objeto no outro.

Esse mapeamento em geral é feito colocando a referência no objeto dependente, ou seja, aquele que para existir precisa que o outro exista primeiro. Por exemplo, um Usuário pode existir sem um Aluno existir (professor), mas o objeto Aluno necessita da existência de um Usuário. É considerada uma boa prática não criar relações bidirecionais.

10.6.1. Um para Um – OneToOne

Esse tipo de relação conecta dois objetos de maneira que o objeto pai tenha apenas um do objeto filho.

Exemplo: Vamos imaginar que no nosso sistema, o Coordenador deve estar vinculado a um único Curso. Para isso, como Curso só irá existir se estiver vinculado a um Coordenador, teremos uma chave estrangeira na sua tabela.

```
...
class Curso(models.Model):
    sigla = models.CharField(unique=True, max_length=5)
    nome = models.CharField(unique=True, max_length=50)
    coordenador = models.OneToOneField(
        "contas.Coordenador",
        null=True # Se o modelo já existia antes
    )
class Coordenador(Usuario):
    sala = models.CharField("Sala", max_length=3)
...
```

10.4 Exemplo Um-para-Um

O uso do relacionamento Um para Um garante que você não possa fazer com que o coordenador tenha mais do que um curso. Ao obter o curso, o objeto resultante no atributo coordenador será do tipo coordenador mesmo (e não uma lista).

10.6.2. Muitos para Um – Many to One

Esse tipo de relação conecta dois objetos de maneira que o objeto pai tenha vários objetos filhos. Do ponto de vista do banco de dados, a modelagem é similar (chave estrangeira na entidade fraca). Exemplo: Uma Disciplina pode ser oferecida em vários anos e semestre (DisciplinaOfertada), mas apenas uma vez por semestre e ano.

```
...
class DisciplinaOfertada(models.Model):
    ano = models.SmallIntegerField("Ano")
    semestre = models.CharField("Semestre", max_length=1)
    disciplina = models.ForeignKey(
        Disciplina,
        null=True # Se o modelo já existia antes
    )
class Disciplina(Usuario):
    nome = models.CharField("Nome")
...
```

10. 5 Exemplo de Muitos para um

Para garantir o tipo Muitos para Um, usamos o campo ForeignKey, passando o nome do modelo a ser ligado. O uso do relacionamento Muitos para Um garante que uma disciplina possa ter várias ofertas. Ao obter a oferta da disciplina, o objeto resultante no atributo disciplina será do tipo coordenador mesmo (e não uma lista).

10.6.3. Muitos para Muitos – Many to Many

Esse tipo de relação conecta dois objetos de maneira que o objeto pai tenha vários objetos filhos e os filhos tenham vários pais. Em geral é representado por uma tabela de associação.

Exemplo: Um aluno pode se matricular em várias Turmas, assim como uma Turma possui vários alunos. Para isso se usa uma tabela associativa (Matrícula)

```
...
class Turma(models.Model):
    disciplina_ofertada = models.ForeignKey(DisciplinaOfertada)
    identificador = models.CharField(max_length=1)
    turno = models.CharField(max_length=15, blank=True, null=True)
    professor = models.ForeignKey("contas.Professor", blank=True, null=True)
    alunos = models.ManyToManyField(
        "contas.Aluno",
        db_table="MATRICULA"
    )
class Aluno(Usuario):
...

```

10. 6 Exemplo de Muitos para Muitos

O uso do relacionamento Muitos para Muitos garante que um aluno possa ter várias turmas e uma turma tenha vários alunos. Ao obter a turma, o objeto resultante no atributo alunos será uma lista de objetos do tipo aluno. Lembrando que primeiro ambas as entidades devem existir (inseridas no banco) antes de serem associadas.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O Django permite a conexão com diversos dos SGBD's mais famosos.
- O ORM do Django permite um mapeamento de classes para tableas e vice-versa.
- O mapeamento ORM permite mapear as relações entre as entidades (um para um, um para muitos e muitos para muitos).

Capítulo 11

Autenticação e Autorização

Django

- Introdução;
- Usuários no Django;
- Login e Logout;
- Bloqueando o acesso.

11.1. Introdução

Nem tudo na internet é público. Precisamos de maneiras efetivas de proteger os dados na web, com um sistema de autenticação e autorização, que permita bloquear certos caminhos para certos usuários.

11.2. Usuários no Django

O Django Admin já possui uma tela de login, mas ela não deve ser usada para usuários comuns, por exemplo, clientes da sua loja. Para isso precisamos criar uma outra maneira de autenticar o nosso usuário (e também para sair do site ou desfazer o login).

Apesar de haverem diversas formas de login (senha, token, etc.), todas possuem a mesma lógica:

- usuário manda um identificador (ex: nome) e um verificador (ex: senha).
- sistema compara com o que existe na base e volta True caso esteja correto ou False caso contrário.

O Django já possui no módulo `django.contrib.auth` a base da implementação de autenticação que precisamos, basta alterarmos alguns detalhes.

Uma das questões é o modelo de usuário. O modelo do Django pode não possuir tudo que precisamos para os nossos usuários. Dessa forma, seria bom estender esse modelo para conter os parâmetros que vamos precisar.

```

from django.contrib.auth.models import AbstractBaseUser, UserManager
ALUNO = 'A'
PROFESSOR = 'P'
COORDENADOR = 'C'
PERFIS = (
    (ALUNO, 'Aluno'),
    (PROFESSOR, 'Professor'),
    (COORDENADOR, 'Coordenador')
)
class Usuario(AbstractBaseUser):
    ra = models.IntegerField("RA", unique=True)
    nome = models.CharField("Nome", max_length=100, blank=True)
    email = models.EmailField("E-Mail", unique=True)
    ativo = models.BooleanField("Ativo", default=True)
    perfil = models.CharField("Perfil", max_length=1, choices=PERFIS)
    def __str__(self):
        return self.nome

```

11.1 Exemplo de modelo de usuário

Para funcionar como um modelo de autenticação, precisamos configurar mais algumas coisas: No modelo devemos colocar outras configurações:

- Uma propriedade definindo qual campo o Django usará como username (login): USERNAME_FIELD = "ra"
- O Objects do usuário não pode ser o comum, vamos usar um próprio para usuários: objects = UsuarioManager()
- O modelo precisa de dois métodos para visualização: um para nome completo e outro para nome curto: get_full_name(self) e def get_short_name(self)
- Precisamos por último indicar os campos obrigatórios: REQUIRED_FIELDS = ["email", "nome"]

A classe UsuarioManager precisa ser modificada para aceitar outro tipo de username (ex: RA).

```
class UsuarioManager(BaseUserManager):
    use_in_migrations = True

    def _create_user(self, ra, password, **extra_fields):
        if not ra:
            raise ValueError('RA precisa ser preenchido')
        user = self.model(ra=ra, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_user(self, ra, password=None, **extra_fields):
        return self._create_user(ra, password, **extra_fields)

    def create_superuser(self, ra, password, **extra_fields):
        return self._create_user(ra, password, **extra_fields)
```

11.2 Exemplo de UserManager

No arquivo settings.py devemos adicionar:

Novo modelo de autenticação: AUTH_USER_MODEL = "contas.Usuario"

- A url de login (para redirecionamentos) LOGIN_URL = 'login'
- A url para onde ir quando o login der certo LOGIN_REDIRECT_URL = 'index'
- A url de logout LOGOUT_URL = 'logout'

Para funcionar agora, precisamos migrar o novo modelo para o BD. Como já temos uma tabela de usuário lá (padrão), o melhor é remover os modelos e mandar inserir novamente (com makemigrations e migrate). Depois de migrado o site, precisamos colocar a view de login e o mecanismo de logout.

11.3. Login e Logout

No arquivo `urls.py`, vamos importar as `views login` e `logout` do Django:

`from django.contrib.auth.views import login, logout`. Vamos criar as `urls 'entrar'` e `'sair'` para ambas as `views`

- `url(r'^entrar/', login, {'template_name': 'login.html'})`,
- `url(r'^sair/', logout, {'next_page': 'index'})`,

Para a `view login` temos uma opção adicional de passar o `template` que vamos usar para o `login` (veremos a seguir). Para a `view logout` precisamos dizer para onde a `view` irá ao fazer o `logout` (nesse caso o `index`)

Para verificar que estamos logados, no `base.html` podemos adicionar o seguinte trecho no `menu`: `{% if user.is_authenticated %} <p>Entrou!</p> {% else %} <p>Não Entrou! </p> {% endif %}`

Assim temos um `menu` condicional se o usuário está ou não logado. Note o uso da `template tag` `{% url 'login' %}`, ela converte automaticamente a `URL` com o nome passado (nesse caso `login`) no seu arquivo `urls.py`.

11.4. Bloqueando acesso

Agora que já temos uma autenticação funcionando no sistema, devemos fazer uma maneira de bloquear algumas `views` para determinados usuários. O bloqueio vai acontecer em dois níveis:

- `Views` que precisam que o usuário esteja logado.
- `Views` que precisam que o usuário tenha um determinado perfil.
- Para executar essa ação, vamos utilizar dois `decorators` do Django

Vamos supor duas `Views`: uma apenas para professores e outra apenas para alunos. Primeiro precisamos bloquear essas `views` para usuários logados. Para isso vamos usar o `decorator @login_required`.

```
from django.contrib.auth.decorators import login_required
@login_required(login_url='/entrar')
def aluno(request):
    return render(request, "aluno.html")
@login_required(login_url='/entrar')
def professor(request):
    return render(request, "professor.html")
```

11.3 Exemplo de `@login_required`

Agora para bloquear views para usuários específicos vamos precisar de funções que verifiquem as nossas condições. Essas funções recebem o usuário como parâmetro de entrada e testem o que é necessário (nesse caso se o perfil está certo). Com essas duas funções prontas, podemos usar o decorator `@user_passes_test`

```
from django.contrib.auth.decorators import login_required, user_
passes_test
# Funções de teste...
@login_required(login_url='/entrar')
@user_passes_test(
    checa_aluno,
    login_url='/?error=acesso',
    redirect_field_name=None
)
def aluno(request):
    return render(request,"aluno.html")
@user_passes_test(
    checa_professor,
    login_url='/?error=acesso',
    redirect_field_name=None
)
@login_required(login_url='/entrar')
def professor(request):
    return render(request,"professor.html")
```

11.4 Exemplo de `user_passes_test`

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Nativamente o Django já possui um sistema de usuários, mas podemos customizá-lo facilmente.
- A lógica do Login e do Logout já está pronta para uso no Django.
- O Django permite a proteção das suas views com apenas alguns decorators.

Capítulo 12

Requisições AJAX

- Introdução;
- AJAX;
- Como usar;
- JSON

12.1. Introdução

Na internet moderna, são necessárias muitas informações para se montar uma página. Dessa maneira, o jeito clássico de retornar listas de informações podem não ser muito eficientes.

Nesse caso queremos buscar informações no servidor de uma maneira “por baixo dos panos”, sem que isso trave nosso navegador, ou seja, de maneira assíncrona.

12.2. AJAX

AJAX é um acrônimo para Asynchronous JavaScript And XML. AJAX é uma maneira de deixar o JavaScript fazer uma requisição HTTP para o servidor, no lugar do navegador.

Requisições AJAX são assíncronas por natureza, ou seja, não bloqueiam a execução do navegador. Elas não recarregam a página como o GET e o POST do navegador.

Mesmo com o X no nome, não é necessário usar o XML para passar dados.

A maneira mais comum de passar dados via AJAX é por JSON (JavaScript Object Notation - veremos adiante). Não é tecnologia, é especificação:

- Dispara um objeto XMLHttpRequest (o mesmo que o navegador usa) para o servidor.
- JavaScript recebe a resposta e já interpreta os dados.

12.3. AJAX

Para usar o AJAX no JavaScript, devem ser feitos os seguintes passos:

- JavaScript prepara a requisição.
- Abre-se a requisição para o servidor com o método, URL e um condicional de sincronidade.
- Ao configurar tudo que for necessário, basta chamar a função send para enviar a requisição.

```
var xhttp = new XMLHttpRequest();
xhttp.open(METODO, URL, ASSINCRONO)
// Um GET assíncrono para 'buscar'
xhttp.open("GET", "buscar", true)
// Um POST síncrono para 'buscar'
xhttp.open("POST", "buscar", false)
xhttp.send();
```

11. 4 Exemplo de user_passes_test

Para requisições GET, os parâmetros vão na URL ao abrir a requisição (ex: xhttp.open("GET", "buscar?nome=Yuri&idade=30", true)). Para requisições POST, precisamos serializar as informações e colocar como parâmetro do send, informando como o servidor deve entender os parâmetros (xhttp.send("nome=Yuri&idade=30")).

O terceiro parâmetro do open identifica se a chamada será assíncrona ou síncrona. Chamadas síncronas: durante a execução da requisição, todo o restante do código JavaScript esperará a requisição retornar antes de continuar. Chamadas assíncronas deixam o restante do JavaScript continuar a sua execução enquanto faz a requisição ao servidor. Como executar algo ao acabar então?

Eventos! O XMLHttpRequest possui um evento que avisa quando a requisição muda de estado: o onreadystatechange

Esse evento é disparado toda vez que a requisição muda de estado. O estado de uma requisição é dado pela propriedade readyState da requisição (xhttp.readyState):

- 0: Requisição ainda não inicializou
- 1: Conexão com o servidor estabelecida.
- 2: Requisição recebida pelo servidor.
- 3: Requisição sendo processada.
- 4: Requisição terminada e resposta recebida

```
xhttp.open("GET", "buscar?nome=Yuri&idade=30", false)
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        alert("Acabou!");
    }
};
xhttp.send();
```

12. 2 Exemplo de controle por eventos

O objeto XMLHttpRequest possui as seguintes propriedades:

- readyState: Estado da requisição.
- status: Código HTTP (200, 404, 500 etc.).
- statusText: Texto do status (ok ou algum erro).
- .responseText: se não houve erro, a resposta em formato de texto (HTML,JSON).

12.4. JSON

Por mais que o AJAX tenha sido criado para utilizar o XML, hoje o formato mais utilizado na Web é o JSON - JavaScript Object Notation. O JSON é uma notação para objetos genéricos do JavaScript, feita com o objetivo de ser uma formatação leve para troca de dados entre servidores.

O formato JSON é muito similar aos dicionários do Python. Fundamentalmente é composto por uma sequência de pares chave-valor separados por vírgula.

```
{  
    "nome" : "Yuri",  
    "idade" : 30  
}
```

12.3 Exemplo básico de um JSON

As chaves dos valores podem ou não estar com aspas. Ambos os próximos JSON são válidos. Preferencialmente utilizem aspas duplas nas chaves. Nas chaves podemos ter apenas Strings/textos, mas para os valores podemos ter quaisquer tipos válidos do JavaScript.

Para utilizar um objeto JSON, podemos colocá-lo dentro de uma variável no JavaScript. Para acessar uma propriedade do JSON podemos usar o operador ponto '.', ou usar o operador de chaves '[']. Ambos usam o nome da propriedade que queremos (o operador chaves precisa do nome como String, entre aspas).

```
var agenda = {  
    "dono" : "Yuri Dirickson",  
    "telefones" : [123456, 987654],  
    "ligar" : function(contato) {  
        alert("Ligando para "+contato);  
    }  
}
```

12.4 JSON em variável

12.5. Enviando JSON pelo AJAX

Para enviar um JSON via POST, podemos montar um objeto JSON com os dados e transformá-lo em String. Imagine um formulário de login: Ao invés de usarmos a submissão via formulário comum, vamos interceptar essa requisição e fazer via AJAX. A ideia é logar o usuário e avisar ele sem recarregar a página, mudando uma região com o nome do usuário..

```
<form method="POST" action="logar" id="form1">
  Usuário: <input name="usuario" type="text" />
  Senha: <input name="senha" type="password" />
  <input type="submit" value="Enviar" />
</form>
```

12. 5 Exemplo de formulário de login

Para interceptar a submissão do formulário, vamos usar o evento onsubmit e construir o nosso JSON com os dados do formulário.

```
var form = document.getElementById("form1")
form.onsubmit = function(event){
event.preventDefault(); // Mata o evento padrão
  var dados = { } //Cria um JSON vazio, podemos usar o new Object()
  for (var i = 0; i < form.elements.length; i++){
    var input= form.elements[i];
    if(input.name){
      dados[input.name] = input.value;
    }
  }
  // Constrói a requisição
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange(... fazer algo ...); // Veremos adiante
  xhr.open(form.method, form.action, true); // Abre a requisição
  // Define o tipo de dados como JSON e coloca o CSRF token no cabeçalho
  xhr.setRequestHeader("X-CSRFToken", dados.csrfmiddlewaretoken);
  xhr.setRequestHeader("Content-Type", "application/json; charset=UTF-8");
  // Envia configurando o JSON como uma String/serializando
  xhr.send(JSON.stringify(dados));
}
```

12. 6 Exemplo de envio de json por formulário

Teste a requisição e veja o console do navegador (na parte network), veja como os dados foram enviados. O utilitário JSON do JavaScript possui algumas funções para lidar com o formato JSON. Uma delas é o stringify, que converte um objeto JSON em uma String.

12.6. Recebendo o JSON no Servidor

Cada linguagem de programação tem a sua forma de entender o JSON, usando outras bibliotecas para traduzi-lo. Lembrando que o JSON é um formato específico, mas seus dados são enviados como String.

No Django, o mais comum é traduzirmos o JSON para um dicionário. Para fazer isso usamos o utilitário `json` do Django. A lógica fica:

- Primeiro usamos a função `json.loads` para traduzir uma String no formato JSON para um dicionário do Python.
- Fazemos a lógica qualquer de login e montamos uma resposta com os dados do usuário logado (`print`).
- Para converter novamente o dicionário para a String no formato JSON, usamos a função `json.dumps` (precisamos setar o `content_type` para `application/json`, pois ele sempre espera voltar HTML nessas requisições).

```
import json

def login(request):
    login = json.loads(request.body.decode("utf-8"))
    # faz a lógica de login, aqui apenas uma impressão
    print(login)
    responseData = { # Cria a resposta com os dados do usuário logado
        "nome": "Yuri",
        "email" : "yuri@email.com"
    }
    return HttpResponse(
        json.dumps(responseData),
        content_type="application/json"
    )
```

12.7 Exemplo de chamada com json

Depois do processamento no Django, retornamos novamente um JSON para o JavaScript, como trabalhar com ele?

Dentro da evento `onreadystatechange` da requisição, esse JSON estará na propriedade `responseText`. Quando ele retornar (status 200 e `readyState` 4), podemos traduzi-lo para um objeto JSON e usar normalmente para atualizar a tela.

12.7. Desvantagens do AJAX

Vimos a principal vantagem do AJAX: chamar o servidor sem travar a tela do usuário. Mas como nem tudo é alegria, aqui vão algumas desvantagens para se levar em conta:

- Retrocompatibilidade: navegadores antigos (IE antes do 6) não possuem essa funcionalidade completa.
- Limites do HTTP 1.1: se tudo for ajax no seu sistema, você pode esbarrar nos limites de requisição do HTTP (2 por domínio).
- Assíncrono, mas nem tanto: Nem todos os navegadores operam da mesma forma com chamadas assíncronas, em específico o Safari tem muitos bug's sem resolução ainda.
- Muito código: Você não precisa escrever um código para receber um HTML no navegador, mas cada AJAX precisa ser re-escrito.

Seus usos mais comuns:

- Frameworks para construir SPA - Single Page Applications - usam AJAX para tudo (Ex: Angular e React).
- Google Suggestion: campos que tentam autocompletar o que o usuário está digitando com textos que estão no banco (veja também o buscar dos ecommerces).
- Atualização de carrinho sem sair da página: alguns ecommerces você pode incluir um produto no carrinho e continuar sua navegação normalmente, mas a figura do carrinho é atualizada.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O javascript puro consegue fazer chamadas assíncronas através do objeto XMLHttpRequest.
- Devido a natureza assíncrona do AJAX, ele deve ser controlado através de eventos de execução.
- Mesmo com o X na sigla, a maneira mais usual e simples de enviar informações via AJAX é usando o JSON.