


# CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

## LEITURA E ESTRUTURAÇÃO DE DOCUMENTOS - PARTE II

Prof. Anderson França

# MATRIZ CURRICULAR

Essa disciplina explora conceitos fundamentais da ciência de dados, incluindo técnicas de coleta, limpeza e análise de dados. Serão utilizados **algoritmos de aprendizado de máquina**, como regressão, árvores de decisão e redes neurais, aplicados à previsão e classificação. Além disso, serão utilizados métodos de automação de processos utilizando ferramentas modernas e técnicas avançadas para otimizar e escalonar soluções analíticas.



1. Introdução à análise e ciência de dados	44
2. Estatística aplicada I – Cultura orientada a dados e transformação digital	90
3. Estatística aplicada II – Ciência de dados, aprendizado de máquinas e otimização	136
4. <i>Deep learning</i> e inteligência artificial	50
5. Inteligência artificial aplicada	80

## OBJETIVO DA AULA

O objetivo desta aula é capacitar os alunos a identificar limitações no uso de técnicas tradicionais de scraping em páginas com carregamento dinâmico e aplicar soluções baseadas em automação de navegador com Selenium para acessar e extrair informações estruturadas de conteúdos HTML renderizados por JavaScript.

Ao final da aula, os alunos serão capazes de:

- Reconhecer as diferenças entre páginas estáticas e dinâmicas, entendendo por que certas páginas não funcionam com requests e BeautifulSoup.
- Utilizar o Selenium para simular a navegação em páginas web, controlando o navegador de forma automatizada.
- Configurar corretamente o WebDriver e compreender sua relação com o navegador instalado.
- Acessar o conteúdo de páginas como o Diário Oficial da União (DOU), renderizadas dinamicamente, e extrair o HTML completo após a execução do JavaScript.
- Integrar o Selenium com BeautifulSoup para localizar, extrair e estruturar elementos do conteúdo carregado.
- Organizar os dados extraídos em listas ou dicionários, facilitando sua posterior análise ou armazenamento.



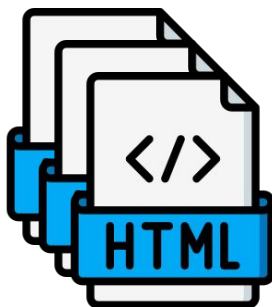
# CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

## RESUMO

# DOCUMENTOS DO DIA-A-DIA



PDF com muitas  
páginas



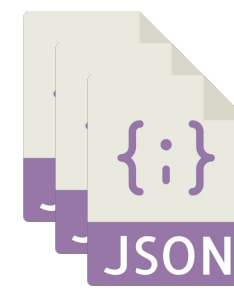
Parecer publicado  
em páginas *web*



Documentos  
criados no setor



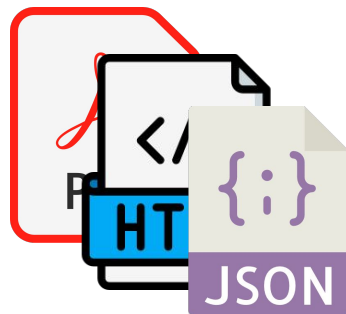
Logs de sistemas e  
dados de auditoria



Respostas de apis e  
sistemas internos

---

Todos são diferentes e  
com o mesmo desafio:



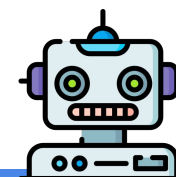
Não são dados  
**AINDA**

# O QUE PRECISAMOS EXTRAIR?



## Modelos Estatísticos e Aprendizado de Máquinas

- Colunas
- Variáveis
- Datas
- Categorias



## Modelos de Linguagem (LLMs)

- Texto Organizado
- Seções Claras
- Contexto Bem Definido

Lembrando:

**Garbage in, Garbage Out**

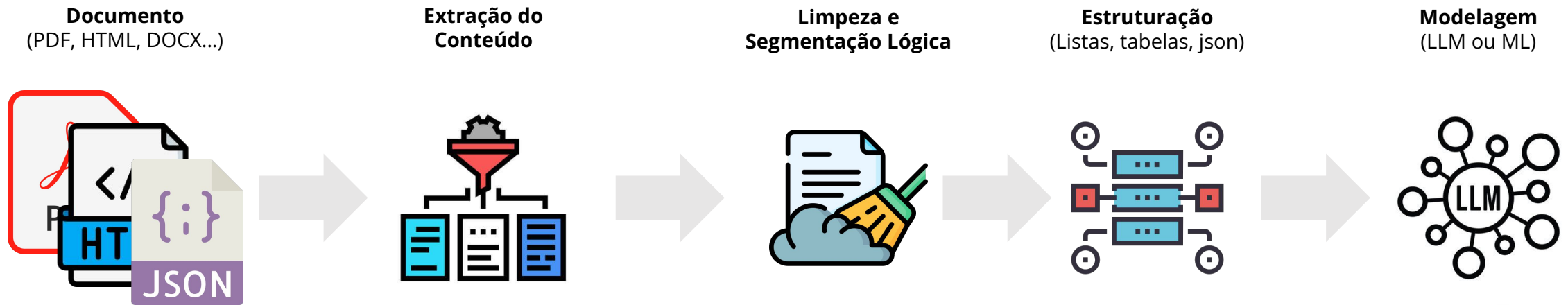
Se a entrada é ruim, o resultado será ruim também



# PIPELINE DE TRANSFORMAÇÃO

Modelos de IA e análises estatísticas exigem entradas organizadas. Documentos em PDF, HTML ou Word raramente vêm prontos para isso.

O processo de transformação envolve etapas bem definidas: extrair, limpar, estruturar e só então modelar.



# INTERPRETAR, ESTRUTURAR E TRANSFORMAR

## Ler documentos não é o mesmo que extrair dados

Entender a estrutura de um arquivo é o primeiro passo para qualquer análise automatizada. Cada documento carrega uma lógica própria, que é explícita para o leitor humano, mas invisível para algoritmos.

Transformar esse conteúdo em dados úteis exige conhecimento técnico, decisões contextuais e ferramentas adequadas.

A partir de agora, vamos explorar os diferentes formatos que vamos encontrar no dia a dia e os desafios específicos que cada um deles impõe ao processo de estruturação.







# DIFICULDADE GERAIS

Apesar de serem estruturas completamente diferentes, as principais dificuldades são:

- Separar conteúdo útil de metadados visuais (headers, rodapés, numeração)
- Manter contexto ao dividir por parágrafos ou seções
- Tratar quebras de linha, espaçamento e símbolos especiais
- Preservar o significado do conteúdo ao “quebrar” o documento em partes menores

**Exemplo:** Representações diferentes para um mesmo parecer técnico

 <b>PDF</b>	 <b>HTML</b>	 <b>DOCX</b>	 <b>CSV</b>
O parecer está em duas colunas, com logotipo institucional no cabeçalho, número do processo no rodapé e várias quebras de página. Visualmente bonito, mas difícil de extrair com precisão.	O mesmo parecer foi publicado em um portal oficial. O texto aparece em blocos divididos por <code>&lt;div&gt;</code> e <code>&lt;p&gt;</code> , e parte do conteúdo (como anexos) é carregado via JavaScript. Requer tratamento para eliminar menus, rodapés e scripts.	O documento original foi escrito no Word. Possui estilos aplicados nos títulos, sumário automático e tabelas organizadas. É o formato mais limpo para extrair, desde que não tenha sido convertido de forma inconsistente.	Um resumo do parecer foi exportado para CSV com colunas como: "Data", "Órgão", "Responsável", "Resumo". Ideal para consulta rápida ou cruzamento com outros dados, mas sem a riqueza de contexto do texto completo.

# ENTENDER O FORMATO É O PRIMEIRO DESAFIO

Antes de qualquer extração, é essencial reconhecer o tipo de documento que estamos trabalhando.

Cada formato tem suas regras, limitações e armadilhas, e isso afeta diretamente a qualidade dos dados que vamos gerar.

Saber identificar e tratar essas diferenças é o que garante que, lá na frente, a análise ou modelo de IA esteja baseado em uma estrutura confiável.



**O formato define estratégia.**

**Não existe extração automática sem  
entendimento prévio do problema.**

# ESTRATÉGIA DEPENDE DO FORMATO

**Nem todo documento pode ser lido da mesma forma.** A estratégia de extração depende do tipo de arquivo e de como o conteúdo está armazenado:

- texto real,
- imagem,
- marcações HTML ou
- estrutura de parágrafos.

Saber escolher a ferramenta certa é o primeiro passo para acessar a informação com precisão.

Formato	Ferramenta Indicada	Tipo de Conteúdo
PDF	pdfplumber, PyMuPDF	Pode ser texto ou imagem
HTML	BeautifulSoup, lxml	Texto com marcação por tags
DOCX	python-docx	Texto estruturado
Escaneado	Tesseract, EasyOCR	Imagem (precisa de OCR)



# LEITURA DE PDF

Para esse exemplo, vamos selecionar um [parecer técnico nº 13, de 2024](#) – Definição da segurança de uso da substância Monoetilenoglicol. (selecionado de forma aleatória)

## Características do Documento:

- Documento oficial da Anvisa em formato PDF.
- Contém seções como introdução, fundamentação técnica e conclusão.
- Possui formatação padronizada, facilitando a extração de texto.

## Informações que serão extraídas:

- Número do Processo
- Substâncias Citadas
- Seção da Conclusão
- Responsável Técnico



### **PARECER Nº 13/2024/SEI/GGCOS/DIRE3/ANVISA**

Processo nº 25351.903963/2022-21  
Interessado: Gerência-Geral de Cosméticos e Saneantes (GGCOS)  
Assunto: Avaliação de Segurança da Substância Monoetilenoglicol -  
Nomenclatura Internacional de Ingrediente Cosmético (INCI):  
Glycol (CAS 107-21-1)

Definição da segurança de uso da substância Monoetilenoglicol - INCI: Glycol (CAS 107-21-1) em produtos de higiene pessoal, cosméticos e perfumes, especialmente em produtos que entram em contato com a mucosa bucal

### **1. Relatório**

A Gerência de Hemo e Biovigilância e Vigilância Pós-Uso de Alimentos, Cosméticos e Produtos Saneantes (GHBIO) da Anvisa foi comunicada em 05/09/2022 pelo Ministério da Agricultura, Pecuária e Abastecimento - MAPA, de investigação envolvendo intoxicação e óbito de animais de companhia após ingestão de petiscos próprios para alimentação animal que apresentaram contaminação do ingrediente Propilenoglicol com Etilenoglicol.

A Gerência de Inspeção e Fiscalização Sanitária de Alimentos, Saneantes e Cosméticos da Anvisa abriu dossiê de investigação, inicialmente para apurar se há risco para a saúde humana e, caso necessário, tomar as medidas sanitárias cabíveis; visto ser imprescindível a verificação da rastreabilidade dos insumos contaminados, para avaliar se o produto foi utilizado por empresa do setor de alimentos de uso humano.

# PDFPLUMBER

Agora que já temos a biblioteca carregada, o próximo passo é **abrir o arquivo PDF** que queremos analisar.

Usamos a estrutura **with** para garantir que o arquivo seja aberto corretamente e fechado automaticamente ao final.

Dentro desse bloco, podemos acessar as páginas do documento e extrair o texto de forma segura e organizada.

```
# Carregar apenas uma página do documento
with pdfplumber.open(caminho) as pdf:
    texto_pag1 = pdf.pages[0].extract_text()
    print(texto_pag1)
```



o `extract_text()` retorna o conteúdo da página como uma string contínua.

```
# Carregar todas as páginas do documento
texto_completo = ""
with pdfplumber.open(caminho) as pdf:
    for pagina in pdf.pages:
        texto_completo += pagina.extract_text(layout=True) + "\n"

print(texto_completo[:1000]) # visualizar primeiros 1000 caracteres
```



Quando usamos esse laço para juntar o conteúdo de todas as páginas, estamos transformando um documento visual em uma **única string contínua de texto**.

# PyMuPDF (fitz)

O PyMuPDF permite extrair o conteúdo da página em formato de dicionário estruturado, com metadados sobre blocos de texto, coordenadas, estilo e ordem de leitura.

```
import pymupdf # pymupdf

# Abre o PDF
doc = pymupdf.open(caminho)

# Lê o texto da primeira página
pagina = doc[0]
texto = pagina.get_text()

print(texto[:1000]) # primeiros 1000 caracteres
```

A base do PyMuPDF é uma engine chamada **MuPDF** (escrita em C++)



O módulo Python foi originalmente chamado de fitz, em homenagem ao executável fitz.exe da versão C++

O nome **“fitz”** foi mantido por compatibilidade, mesmo após a criação do pacote pymupdf

Portanto, pode ser que ao pesquisar sobre essa biblioteca ou usar ferramentas de LLM para escrever o código, podemos nos deparar com **import fitz**.



# PREPARANDO O TEXTO EXTRAÍDO

Antes de estruturar ou analisar, é essencial fazer uma etapa de limpeza e padronização do conteúdo, pois, o texto extraído de um PDF raramente vem pronto para uso. Ele geralmente apresenta **ruídos estruturais** como:

- Quebras de linha fora de lugar
- Espaços duplicados
- Cabeçalhos repetidos em todas as páginas
- Palavras cortadas ou desalinhadas
- Resíduos de formatação visual que não fazem sentido no texto corrido

Ação	Objetivo	Função em Python
Remover <code>\n</code> soltos	Unir linhas que deveriam ser parágrafos	<code>str.replace("\n", " ")</code>
Remover espaços duplos	Corrigir quebras e layout	<code>str.replace(" ", " ")</code>
Padronizar caixa	Facilitar buscas e comparação	<code>.lower()</code> ou <code>.title()</code>
Corrigir acentuação (opcional)	Padronizar internacionalmente	<code>unidecode()</code>
Remover headers/repetições	Deixar texto mais limpo para análise	Regex ou regras manuais

# PRÉ-PROCESSAMENTO DE DADOS

Após extrair o conteúdo do documento, o texto ainda contém **quebras artificiais, palavras fragmentadas e espaçamento irregular**, que dificulta qualquer análise posterior.

Este pré-processamento inicial tem como objetivo **unificar o texto**, corrigir pequenas distorções causadas pela extração do PDF e deixá-lo em um formato mais **limpo, contínuo e padronizado** e pronto para ser segmentado e estruturado.

```
import re

# Remove quebras de linha quebradas
texto_limpo = texto_completo.replace("\n", " ")

# Remove múltiplos espaços
texto_limpo = re.sub(r" {2,}", " ", texto_limpo)

# Remove palavras hifenizadas
texto_limpo = re.sub(r"-\\s+", "", texto_limpo)

# Remove espaços extras nas pontas
texto_limpo = texto_limpo.strip()

# Visualizar texto
print(texto_limpo[:2000])
```

# PREPARANDO O TEXTO EXTRAÍDO

Depois da limpeza, o próximo passo é **organizar o conteúdo** do documento em partes significativas, como seções, campos ou registros. Essa estruturação permite que o texto bruto se torne um **conjunto de dados reutilizável**, seja em forma de dicionário, tabela ou JSON.

Em muitos documentos, precisamos **identificar padrões específicos** dentro de grandes blocos de texto, como:

- Números de processo
- Datas
- Números CAS
- Palavras-chave

Esses elementos não seguem exatamente a mesma frase, mas seguem um padrão de escrita.

As expressões regulares (ou **regex**) são uma ferramenta que permite **encontrar padrões textuais com precisão**.



# EXPRESSÕES REGULARES

Vamos começar procurando um número de processo.

**25351.903963/2022-21**

Note que esse processo possui um padrão

**XXXXX.XXXXXXX/XXXX-XX**

```
# Expressão Regular para extrair processo
import re
re.findall(r"\d{5}\.\d{6}/\d{4}-\d{2}", texto_limpo)
```

- `\d{5}` → 5 dígitos
- `\.` → um ponto literal
- `\d{6}` → 6 dígitos
- `/` → barra
- `\d{4}` → 4 dígitos
- `-` → hífen
- `\d{2}` → 2 dígitos

# EXPRESSÕES REGULARES

Expressão regular (ou **regex**) é uma linguagem utilizada para descrever padrões em textos. Com ela, conseguimos **identificar, extrair, validar** ou **substituir** partes de um texto que seguem uma estrutura específica, mesmo quando não sabemos exatamente o conteúdo.

É amplamente usada em programação para tarefas como:

- Encontrar datas, códigos, e-mails, números de documentos
- Validar formatos de entrada (como CPF ou CEP)
- Limpar ou transformar textos automaticamente

Para testar ou aprender mais sobre regex, acesse: [regex101.com](https://regex101.com)

Símbolo	Significado	Exemplo prático
\d	Um dígito (0 a 9)	\d\d\d\d\d\d\d → 12/03/2024
\w	Um caractere de palavra (letra, número, _)	\w+ → qualquer palavra
.	Qualquer caractere (exceto quebra de linha)	a.b → casa, a1b, axb
+	Um ou mais do elemento anterior	\d+ → 1, 123, 2024
*	Zero ou mais do elemento anterior	\w* → palavra ou vazio
{n}	Exatamente n repetições	\d{4} → 2024
{n,m}	De n a m repetições	\d{2,4} → 23, 2024
[...]	Qualquer um dos caracteres entre colchetes	[ABC] → A, B ou C
()	Agrupa uma parte do padrão	(\d{2}/\d{2}/\d{4}) → data completa
		OU lógico
^	Início da linha	^conclusão → linha que começa com "conclusão"
\$	Fim da linha	fim\$ → linha que termina com "fim"

# EXTRAIR INFORMAÇÕES

```
# Expressões regulares
processos = re.findall(r"\d{5}\.\d{6}/\d{4}-\d{2}", texto_limpo)
datas = re.findall(r"\d{2}/\d{2}/\d{4}", texto_limpo)
cas_numbers = re.findall(r"\d{2,7}-\d{2}-\d", texto_limpo)

assinaturas = re.findall( r"documento assinado eletronicamente por (.+?),.*?em (\d{2}/\d{2}/\d{4})", texto_limpo, flags=re.IGNORECASE)

# Se houver assinatura, pega a primeira
nome_assinante = assinaturas[0][0] if assinaturas else None
data_assinatura = assinaturas[0][1] if assinaturas else None

# Organizar em DataFrame
dados_extraidos = pd.DataFrame([{
    "numero_processo": processos[0] if processos else None,
    "datas_mencionadas": ", ".join(datas),
    "numeros_cas": ", ".join(cas_numbers),
    "assinante": nome_assinante,
    "data_assinatura": data_assinatura
}])
```



# EXERCÍCIOS

Você está trabalhando com o conteúdo de mensagens institucionais extraídas de páginas públicas da Anvisa. Entre os textos extraídos, existem **informações úteis misturadas**, como telefones, datas e e-mails de contato.

Vamos utilizar expressões regulares (regex) para extrair todos os **endereços de e-mail válidos**, **número de telefone** e **Data da Reunião**.

```
texto = ""  
Entre em contato com nossa equipe:  
- ana.silva@anvisa.gov.br  
- suporte@empresa.com  
- regulatorio@anvisa.gov.br  
Telefone: (61) 99999-1234  
Data da reunião: 27/09/2024  
Envie dúvidas para: atendimento.anvisa@anvisa.gov.br ou contatar@empresa.org  
""
```

- Use a função `re.findall()`
- Considere que um e-mail válido contém uma sequência de caracteres, um @ e outra sequência (não precisa validar o domínio)
- Imprima a lista de e-mails encontrados
- Imprima a data da reunião
- Imprima o Telefone de Contato

# CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

## HTML PARA WEB SCRAPING



MINISTÉRIO DA  
SAÚDE



# HTML (HyperText Markup Language)

É o que está por trás das páginas da internet, portais e sistemas online. Os textos são divididos por blocos e marcadores, o que ajuda na navegação, mas pode virar um desafio quando o conteúdo está espalhado ou misturado com elementos visuais.

## Características:


- Estrutura marcada por tags (div, p, table)
- Conteúdo pode vir fragmentado por blocos
- Pode conter scripts e elementos interativos
- Facilidade em navegar com BeautifulSoup, mas requer filtragem

## Pontos de Atenção:

- Pode conter ruído: menus, botões, scripts
- Texto pode vir fragmentado em várias tags
- Requer filtros para isolar o conteúdo principal
- Conteúdo pode estar em múltiplas páginas



Site Agência GOV



```
<div class="outer-wrapper">...</div>
<!--/outer-wrapper -->
<footer id="portal-footer-wrapper">...</footer>
...
<!-- Início VLibras Widget --> == $0
<!-- https://vlibras.gov.br/doc/widget/installation/webpageintegration.html
-->
<div vw class="enabled" style="left: initial; right: 0px; top: 50%; bottom:
initial; transform: translateY(calc(-50% - 10px));">...</div>
<script src="https://vlibras.gov.br/app/vlibras-plugin.js"></script>
<script> new window.VLibras.Widget('https://vlibras.gov.br/app'); </script>
<!-- Fim VLibras Widget -->
<footer>...</footer>
<script src="/++plone++ebc.agenciagov.stylesheets/script.js?v3"></script>
<script src="/++plone++ebc.agenciagov.stylesheets/limite.js?v3"></script>
<div id="plone-analytics">...</div>
<iframe allow="join-ad-interest-group" data-tagging-id="G-X0D00CT40G" data-
```

HTML do Site Agência GOV



# LEITURA DE HTML (SIMPLES)

Muitos documentos públicos, como pareceres técnicos, relatórios ou atas, são publicados diretamente em portais governamentais ou sistemas internos via HTML. Isso nos permite acessar o conteúdo de forma mais leve, mas exige filtragem.

## Por que começamos com HTML?

- **É acessível:** basta uma URL ou o código da página
- **Tem estrutura marcada:** com tags como `<p>`, `<div>`, `<table>`, etc.
- **Requer limpeza:** menus, scripts e rodapés poluem o conteúdo
- **É ótimo para treinar leitura semântica de blocos**

```
5 </head>
6 <!-- Type an abbreviat
7 <body>
8 </body>
9 </html>
10
```



# ESTRUTURA BÁSICA DE UMA PÁGINA

Toda página web começa com a estrutura ao lado. Os elementos principais são:

- **<head>**: informações sobre a página (título, configurações, css, javascript)
- **<body>**: onde fica o conteúdo visível e que geralmente estão os dados que queremos (títulos, parágrafos, links, imagens)

Essa estrutura organiza a página em blocos que podem ser acessados pelas nossas ferramentas de scraping.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Minha Página</title>
  </head>
  <body>
    <h1>Título</h1>
    <p>Um parágrafo.</p>
    <a href="https://site.com">Link</a>
  </body>
</html>
```

# PRINCIPAIS ELEMENTOS QUE USAMOS EM SCRAPING

As páginas web são construídas com **HTML**, uma linguagem que organiza o conteúdo por meio de **tags**.

As **tags** funcionam como “etiquetas” que indicam o tipo de informação que está sendo exibida, como texto, imagem, link ou tabela. Elas sempre aparecem entre sinais de menor e maior, como <p> ou <div>, e quase sempre têm uma abertura e um fechamento:

**Por exemplo:**

```
<p>Este é um parágrafo</p>
```

As tags mais comuns em tarefas de *web scraping* são:

Tag	Função	Descrição
<p>	Parágrafo	Define um bloco de texto corrido, como em textos e descrições.
<a>	Link	Cria um link clicável. O endereço está no atributo <b>href</b> .
<div>	Bloco de conteúdo	Agrupa elementos em blocos. Muito usado para organizar o layout.
<h1>, <h2>...	Título e subtítulo	Representam títulos hierárquicos. <b>&lt;h1&gt;</b> é o mais importante.
<img>	Imagem	Insere uma imagem na página. A fonte do arquivo está no atributo src.
<table>	Tabela	Organiza dados em linhas e colunas. Ideal para informações tabulares.

# TAGS ESTRUTURAIS

Nem todo elemento HTML representa texto ou imagem. Algumas tags servem para organizar e agrupar conteúdo. Portanto, quando buscamos uma tag como `<p>`, `<span>` e demais tags, elas vão estar dentro de outros blocos. Essas estruturas ajudam a organizar o layout da página e são muito importantes para tarefas de *scraping*. As principais são:

Tag	Função	Descrição
<code>&lt;div&gt;</code>	Bloco de Conteúdo	Agrupa elementos em seções. Muito usada para estruturar o layout de páginas.
<code>&lt;span&gt;</code>	Destaque em Linha	Agrupa trechos curtos de texto, sem quebrar a linha. Útil para aplicar estilo ou identificar partes específicas.
<code>&lt;section&gt;</code>	Seção temática	Agrupa conteúdo relacionado. Tem significado semântico (ex: uma sessão de artigos).
<code>&lt;article&gt;</code>	Conteúdo Independente	Representa um conteúdo completo e autônomo, como uma notícia ou post.
<code>&lt;header&gt;</code>	Cabeçalho	Define a parte superior de uma seção ou página (título, menu, logo...). Geralmente igual para todas as páginas do mesmo site
<code>&lt;footer&gt;</code>	Rodapé	Define a parte inferior de uma seção ou página (dados de contato, links úteis).

# ATRIBUTOS HTML

As tags HTML podem conter atributos, que trazem informações adicionais sobre o conteúdo.

- Um atributo aparece dentro da tag de abertura.
- Ele sempre segue o formato: **nome="valor"**
- Os atributos ajudam a identificar, descrever ou configurar os elementos da página.

Exemplo:

```
<a href="https://site.com" class="botao">Clique aqui</a>
```

Neste caso:

- **href**: define o destino do link
- **class**: agrupa o elemento em uma categoria para aplicar estilo ou localizar no scraping

Atributo	Descrição
href	Caminho de um link (usado na tag <a>)
src	Caminho de uma imagem (usado no <img>)
alt	Texto alternativo da imagem

Atributo	Descrição
class	Agrupamento de elementos com a mesma função visual/estrutural
id	Identificador único na página
name	Usado em formulários ou campos de entrada



# HTML COM ATRIBUTOS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemplo de Página</title>
  </head>
  <body>
    <!-- Título principal da página -->
    <h1 id="titulo-principal">Bem-vindo ao site da Anvisa</h1>

    <!-- Parágrafo com uma classe -->
    <p class="descricao">Aqui você encontra informações atualizadas sobre saúde pública.</p>

    <!-- Link com atributos -->
    <a href="https://www.gov.br/anvisa" class="botao-link">Acesse o site oficial</a>

    <!-- Imagem com caminho e texto alternativo -->
    

    <!-- Tabela com dados fictícios -->
    <table class="dados">
      <tr>
        <th>Data</th>
        <th>Assunto</th>
      </tr>
      <tr>
        <td>2025-05-06</td>
        <td>Nova regulamentação de medicamentos</td>
      </tr>
    </table>
  </body>
</html>
```

- **id="titulo-principal"** - identifica de forma única o título na página
- **class="descricao"** e **class="botao-link"** - agrupam elementos para estilização ou seleção via scraping
- **href="..."** - define o destino do link
- **src="..."** e **alt="..."** - definem a imagem e o texto alternativo

# HIERARQUIA SIMPLES

```
<body>
  <header>
    <h1>Logotipo da Anvisa</h1>
    <nav>
      <a href="/noticias">Notícias</a>
      <a href="/contato">Contato</a>
    </nav>
  </header>

  <main>
    <section class="lista-noticias">
      <article class="noticia">
        <h2 class="titulo-noticia">Anvisa publica nova regulamentação</h2>
        <p class="data">Publicado em: 06/05/2025</p>
        <div class="conteudo">
          <p>A nova norma trata da venda de canetas emagrecedoras...</p>
          <a href="/noticia-completa">Leia mais</a>
        </div>
      </article>
    </section>
  </main>

  <footer>
    <p>Agência Nacional de Vigilância Sanitária</p>
  </footer>
</body>
```

## Explicação da hierarquia:

- **<body>** contém todo o conteúdo visível.
- Dentro dele, temos três grandes blocos:
  - **<header>** com o menu e logotipo
  - **<main>** com a seção principal de notícias
  - **<footer>** com informações finais
- As notícias estão dentro de **<article>** (bloco semântico), agrupadas por uma **<section>**.
- Dentro do artigo:
  - Título (**<h2>**)
  - Data (**<p>**)
  - Texto da notícia em uma **<div>**
  - Link para mais conteúdo (**<a>**)

# BEAUTIFULSOUP

O BeautifulSoup é uma biblioteca Python usada para extrair e navegar em **dados de páginas HTML e XML**. Ele é leve, simples e extremamente útil para acessar e organizar informações que estão estruturadas na web.

# BeautifulSoup

É uma biblioteca muito útil, pois:

- Permite ler o conteúdo de **qualquer página HTML estática**
- Acessa facilmente tags como `<p>`, `<div>`, `<table>`
- Filtra elementos por classe, id, nome ou estrutura
- Ideal para coletar dados públicos, organizar textos e automatizar consultas

# BEAUTIFULSOUP

Vamos usar a biblioteca **BeautifulSoup** para acessar e extrair o conteúdo textual de uma notícia publicada no site da Anvisa.

O código realiza três etapas:

1. **Faz uma requisição HTTP** para obter o conteúdo HTML da página usando **requests**.
2. **Interpreta o HTML** com o **BeautifulSoup**, permitindo navegar pela estrutura do documento.
3. **Extrai todos os parágrafos (<p>)** da página, limpa os espaços e imprime somente os trechos com texto real.

```
from bs4 import BeautifulSoup
import requests

url = "https://www.gov.br/anvisa/pt-br/assuntos/noticias-anvisa/2025/confira-os-destaques-da-6a-reuniao-publica-da-dicol-de-2025"

resposta = requests.get(url)

soup = BeautifulSoup(resposta.text, "html.parser")
conteudo = soup.find_all("p")

for paragrafo in conteudo:
    texto = paragrafo.get_text(strip=True)
    if texto:
        print(texto)
```



# INSPECIONAR ELEMENTOS

Para fazer scraping com precisão, precisamos encontrar exatamente onde está o dado na página. Para isso, usamos a ferramenta Inspeccionar Elemento do navegador. Para inspecionar o código da página, basta apertar **F12** ou clicar com o **botão direito no elemento da página** e em seguida em **inspecionar elemento**.

**Exemplo:** [Resolução RDC N° 551, de 30 de agosto de 2021](#)

The screenshot displays the 'Diário Oficial da União' website. At the top, there are navigation links: 'VERSÃO CERTIFICADA', 'DIÁRIO COMPLETO', and 'IMPRESSÃO'. The main header features the Brazilian coat of arms and the title 'DIÁRIO OFICIAL DA UNIÃO'. Below this, it states 'Publicado em: 31/08/2021 | Edição: 165 | Seção: 1 | Página: 139'. The main content area is titled 'RESOLUÇÃO RDC N° 551, DE 30 DE AGOSTO DE 2021'. A tooltip for 'p.identifica' shows dimensions '765 x 28.8'. The 'DevTools' 'Elements' panel is open on the right, showing the following HTML structure:

```
<div class="clearfix journal-content-article" data-analytics-asset-id="3416/2891" data-analytics-asset-title="RESOLUÇÃO RDC N° 551, DE 30 DE Agosto DE 2021" data-analytics-asset-type="web-content">
  <div class="dou-modelo">
    <div class="portlet-decorate left-underline"></div>
    <div class="social-media-share border-top border-bottom py-2"></div>
    <ul class="barra-botoes-materia text-center"></ul>
    <article id="materia">
      <div class="row-fluid">
        <div class="cabecalho-dou text-center">
          <div class="brasao-brasil" alt="Cabeçalho - Brasão do Brasil" title="Cabeçalho - Brasão do Brasil"></div>
          <h2 class="cabecalho-titulo-dou">Diário Oficial da União</h2>
        </div>
        <div class="detalhes-dou"></div>
        <div class="texto-dou">
          <p class="identifica">RESOLUÇÃO RDC N° 551, DE 30 DE Agosto DE 2021</p>
          <p class="ementa"></p>
          <p class="dou-paragraph"></p>
          <p class="dou-paragraph"></p>
          <p class="dou-paragraph"></p>
          <p class="dou-paragraph"></p>
          <p class="dou-paragraph">Art. 3º Para efeito desta Resolução são adotadas as seguintes definições:</p>
        </div>
      </div>
    </article>
  </div>
</div>
```

# CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

## EXERCÍCIOS EXTRAÇÃO REQUESTS



# PROBLEMA 1 - CREATINAS

Imagine que surgiu a necessidade de extrair o conteúdo de uma notícia sobre resultados de análise em suplementos publicada no portal da Anvisa para fins de documentação, análise ou arquivamento estruturado.

Nosso objetivo é acessar a seguinte página diretamente pelo Python:

<https://www.gov.br/anvisa/pt-br/assuntos/noticias-anvisa/2025/creatinas-anvisa-divulga-resultados-de-analise-em-suplementos>

A partir dessa URL, vamos desenvolver um código que:

- Acesse a página HTML e identifique apenas os parágrafos relevantes do conteúdo principal
- Limpe o texto extraído, removendo espaços desnecessários ou parágrafos em branco
- Armazene o conteúdo de duas formas:
  - Como uma lista de parágrafos individuais
  - Como um único texto corrido, separado por quebras de parágrafo
- Extrair a tabela, se houver

## Creatinas: Anvisa divulga resultados de análise em suplementos

Resultados de laboratório apontaram diversas incorreções de rotulagem, mas teores dos produtos estavam dentro do esperado.



Agência Gov | Via Anvisa

23/04/2025 16:32



# PROBLEMA 2 - ANTAGONISTAS GLP-1

Temos que estruturar o conteúdo de uma notícia publicada no portal da Anvisa, relacionada à regulamentação de medicamentos utilizados para emagrecimento.

Essa notícia será utilizada em um relatório interno e precisa estar disponível em formato limpo e estruturado, contendo apenas os textos principais da matéria.

Seu objetivo é acessar a seguinte página diretamente com Python:

<https://www.gov.br/anvisa/pt-br/assuntos/noticias-anvisa/2025/canetas-em-agrecedoras-so-poderao-ser-vendidas-com-retencao-de-receita>

A partir dessa URL, vamos desenvolver um código que:

- Acesse a página HTML e identifique apenas os parágrafos relevantes do conteúdo principal
- Limpe o texto extraído, removendo espaços desnecessários ou parágrafos em branco
- Armazene o conteúdo de duas formas:
  - Como uma lista de parágrafos individuais
  - Como um único texto corrido, separado por quebras de parágrafo
- Extrair a tabela, se houver

## Anvisa obriga retenção de receita para venda de canetas como Ozempic

Para agência, retenção é necessária para aumentar o controle do uso

ANDRÉ RICHTER - REPÓRTER DA AGÊNCIA BRASIL

Publicado em 16/04/2025 - 17:40  
Brasília





# PROBLEMA 3 - DOU

Você foi designado para automatizar a extração de publicações recentes do Diário Oficial da União (DOU). O objetivo é incorporar essas informações em sistemas internos de análise e acompanhamento de normativas governamentais.

A partir da seguinte URL:

<https://www.in.gov.br/web/dou/-/portaria-gm/ms-n-6.882-de-22-de-abril-de-2025-627636172>

Desenvolva um código em Python que:

- Acesse o conteúdo da página de leitura do DOU usando **requests**
- Identifique e extraia os **títulos das publicações** exibidas no dia atual
- Extraia, se possível, os **resumos ou trechos principais** associados a cada título
- Armazene essas informações de forma estruturada, como uma **lista de dicionários** contendo:
  - "titulo"
  - "resumo"



## DIÁRIO OFICIAL DA UNIÃO

Publicado em: 06/05/2025 | Edição: 83 | Seção: 1 | Página: 113

Órgão: Ministério da Saúde/Agência Nacional de Saúde Suplementar/Diretoria Colegiada

### DECISÃO DE 31 DE MARÇO DE 2025

A Diretoria Colegiada da AGÊNCIA NACIONAL DE SAÚDE SUPLEMENTAR - ANS, no uso de suas atribuições legais, e tendo em vista o disposto no inciso VI do artigo 10 da Lei nº 9.961, de 28 de janeiro de 2000 em deliberação através da 618ª Reunião de Diretoria Colegiada - DC Ordinária, realizada em 10 de fevereiro de 2025, julgou o seguinte processo administrativo:

# CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

## PREPARAÇÃO DO AMBIENTE



MINISTÉRIO DA  
SAÚDE



# AMBIENTE LOCAL

Para fazer scraping de páginas com conteúdo dinâmico (carregado via JavaScript), precisamos simular um navegador automático, que simula um navegador real. Para essa tarefa, o Colab não vai funcionar de forma apropriada por alguns motivos

- O Colab roda em um servidor remoto, sem acesso direto à interface gráfica de um navegador.
- Precisamos de um navegador **real ou em modo headless**, o que não funciona de forma estável no Colab.
- Recomendação: **rodar localmente**, no computador.

Para isso, vamos precisar usar um **webdriver**.

O WebDriver manipula um navegador nativamente, como um usuário faria, seja localmente ou em uma máquina remota usando o servidor Selenium, marca um salto em termos de automação do navegador.<sup>1</sup>

<sup>1</sup> WebDriver: <https://www.selenium.dev/pt-br/documentation/webdriver/>



# SELENIUM

O **Selenium** é uma ferramenta que permite **automatizar a navegação em sites**, como se fosse um usuário humano interagindo com o navegador.



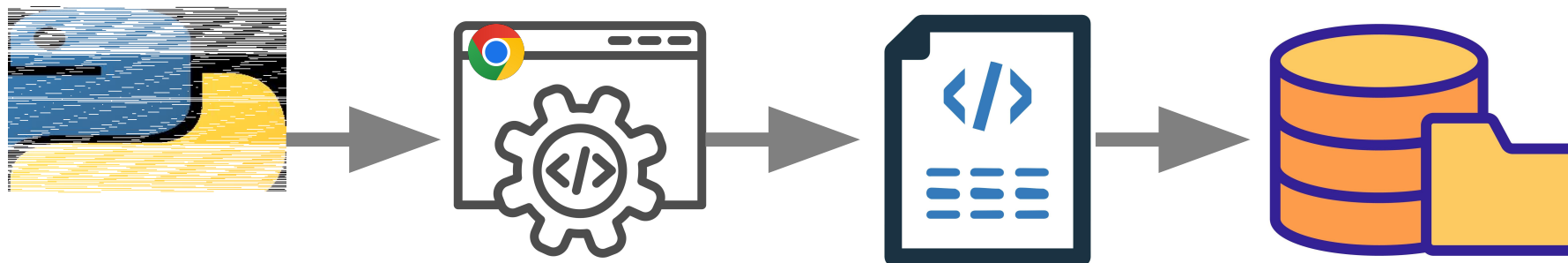
No nosso contexto:

- Carregar páginas que usam **JavaScript** para exibir o conteúdo.
- **Clicar em botões**, rolar a página, preencher formulários e aguardar carregamentos.
- **Capturar textos, links, dados e tabelas** que não aparecem no HTML estático.



# WEBDRIVER

O WebDriver é o componente que controla o navegador via código. Ele atua como um "intérprete" entre o Selenium e o navegador (Chrome, Firefox etc). Cada navegador tem seu próprio WebDriver (ex: Chrome possui o ChromeDriver).



- Escrevemos os comandos em Python para ativar o Selenium.
- O Selenium **controla um navegador real** (como o Chrome).
- O navegador abre, carrega o site, executa os scripts e permite que você extraia o conteúdo já renderizado.

# PREPARANDO O AMBIENTE

Antes de começar a automatizar a navegação em sites, precisamos preparar nosso ambiente com duas ferramentas essenciais:

1. **Selenium:** biblioteca principal para automação de navegador em Python.
2. **webdriver-manager:** ferramenta auxiliar que faz o download automático do ChromeDriver na versão correta, evitando conflitos com o navegador.

## Instalação:

```
!pip install selenium  
!pip install webdriver-manager
```

Isso garante que teremos tudo pronto para:

- Controlar o navegador Chrome via Python
- Evitar problemas de compatibilidade com versões do ChromeDriver
- Começar a abrir páginas, clicar em botões e extrair conteúdo renderizado

# TESTANDO O SELENIUM

Vamos realizar um teste simples para garantir que tudo está funcionando.

O objetivo é abrir automaticamente o navegador, acessar uma página da web, capturar o título da página e, em seguida, fechar o navegador. Esse teste serve como validação do ambiente e demonstra como o Python pode controlar o navegador de forma programada. Se tudo estiver certo, o navegador será aberto e fechado automaticamente, confirmando que você está pronto para seguir com automações mais avançadas.

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager

# Acessa um site qualquer
driver.get("https://www.gov.br/saude/pt-br/composicao/seidigi/rnds")

# Imprime o título da página
print(driver.title)

# Fecha o navegador
driver.quit()
```

O que acontece:

- O WebDriver abre o Chrome.
- Navega até a página especificada (**get()**).
- O Selenium lê o **título da aba**.
- Fecha tudo no final.

# OBSERVAÇÃO IMPORTANTE

```
service = Service(ChromeDriverManager().install())
```

Essa função:

- **Verifica** se a versão correta do ChromeDriver **já está instalada** no cache local (~/.wdm no Linux/Mac ou C:\Users\SeuUsuario\wdm no Windows).
- **Se já estiver instalada, não baixa de novo**, apenas retorna o caminho do executável.
- **Se não estiver, faz o download automático** da versão compatível com o navegador atual.



# OBSERVAÇÃO IMPORTANTE

Em alguns ambientes, o **webdriver-manager** pode falhar ao tentar baixar o ChromeDriver automaticamente, seja por falta de internet, bloqueio de proxy ou restrições da rede. Nesses casos, podemos resolver o problema baixando o ChromeDriver manualmente.

Acesse o site oficial:

<https://googlechromelabs.github.io/chrome-for-testing/>

Baixe a versão do ChromeDriver compatível com o seu navegador e salve o executável na mesma pasta do seu script Python.

```
driver = 'c:\\Users\\caminho\\chromedriver.exe'

service = Service(driver)
driver = webdriver.Chrome(service=service)

# Acessa um site qualquer
driver.get("https://www.gov.br/saude/pt-br/composicao/seidigi/rnds")

# Imprime o título da página
print(driver.title)

# Fecha o navegador
driver.quit()
```

# PROBLEMA 3 - DOU

Agora utilizando o Selenium vamos automatizar a extração de publicações recentes do Diário Oficial da União (DOU). O objetivo é incorporar essas informações em sistemas internos de análise e acompanhamento de normativas governamentais.

A partir da seguinte URL:

<https://www.in.gov.br/web/dou/-/portaria-gm/ms-n-6.882-de-22-de-abril-de-2025-627636172>

Desenvolva um código em Python que:

- Acesse o conteúdo da página de leitura do DOU usando **requests**
- Identifique e extraia os **títulos das publicações** exibidas no dia atual
- Extraia, se possível, os **resumos ou trechos principais** associados a cada título
- Armazene essas informações de forma estruturada, como uma **lista de dicionários** contendo:
  - "titulo"
  - "resumo"



## DIÁRIO OFICIAL DA UNIÃO

Publicado em: 06/05/2025 | Edição: 83 | Seção: 1 | Página: 113

Órgão: Ministério da Saúde/Agência Nacional de Saúde Suplementar/Diretoria Colegiada

### DECISÃO DE 31 DE MARÇO DE 2025

A Diretoria Colegiada da AGÊNCIA NACIONAL DE SAÚDE SUPLEMENTAR - ANS, no uso de suas atribuições legais, e tendo em vista o disposto no inciso VI do artigo 10 da Lei nº 9.961, de 28 de janeiro de 2000 em deliberação através da 618ª Reunião de Diretoria Colegiada - DC Ordinária, realizada em 10 de fevereiro de 2025, julgou o seguinte processo administrativo:

# RESOLVENDO COM SELENIUM

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# Executar o webdriver
service = Service(ChromeDriverManager().install())
driver = webdriver.Chrome(service=service)

# Acessa a página do DOU
driver.get("https://www.in.gov.br/web/dou/-/portaria-gm/ms-n-6.882-de-22-de-abril-de-2025-627636172")

# Aguarda o carregamento do conteúdo
WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.CLASS_NAME, "identifica"))
)

# Captura o HTML completo renderizado
html_renderizado = driver.page_source

# Fecha o navegador
driver.quit()

print(html_renderizado)
```



# CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

## ATIVIDADE PRÁTICA



MINISTÉRIO DA  
SAÚDE





# DESAFIO DA TURMA



# REFERÊNCIAS

- **BeautifulSoup** – Documentation. <https://www.crummy.com/software/BeautifulSoup/bs4/doc.ptbr/>
- **pdfplumber** – API Reference. <https://github.com/jsvine/pdfplumber>
- **PyMuPDF** – Documentation. <https://pymupdf.readthedocs.io/>
- **Docling** – GitHub Repository. <https://github.com/docling-project/docling>
- **REGEX101** – Regex Testing, Explanation and Reference. <https://regex101.com/>
- **Python – re** — Regular expression operations. <https://docs.python.org/3/library/re.html>
- **Requests** – HTTP for Humans. <https://docs.python-requests.org/>
- **Lutz, M. (2013)**. Learning Python (5th Edition). O'Reilly Media.  
<https://www.oreilly.com/library/view/learning-python-5th/9781449355722/>
- **Selenium – Python Documentation**. [https://www.selenium.dev/documentation/webdriver/getting\\_started/](https://www.selenium.dev/documentation/webdriver/getting_started/)
- **WebDriver Manager – Automatize o download do ChromeDriver**. <https://pypi.org/project/webdriver-manager/>
- **WebDriver for Chrome (ChromeDriver) – Downloads e compatibilidade**.  
<https://googlechromelabs.github.io/chrome-for-testing/>



# CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL



MINISTÉRIO DA  
SAÚDE

