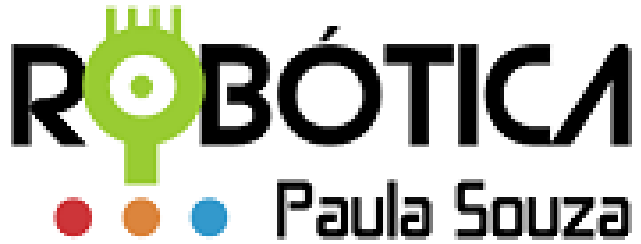
Prof. Me. Jean

Workshop de Web Scraping



O que é Web Scraping?

Web scraping ou Raspagem Web é uma técnica que permite a coleta de informações de websites de forma automática.

Imagine que você deseja reunir dados de várias páginas, como preços de produtos, horários de eventos ou informações sobre notícias. Fazer isso manualmente pode ser muito trabalhoso e demorado.

Web Scraping

- Com o Web Scraping, usamos programas de computador (neste caso o Python) para acessar essas páginas e extrair as informações desejadas. É como ter um assistente virtual que lê as páginas da web para você e coleta os dados que precisa.
- Essa técnica é muito útil para pesquisadores, desenvolvedores e empresas que precisam de grandes quantidades de dados de maneira rápida e eficiente.

ROBÓTICA
Paula Souza

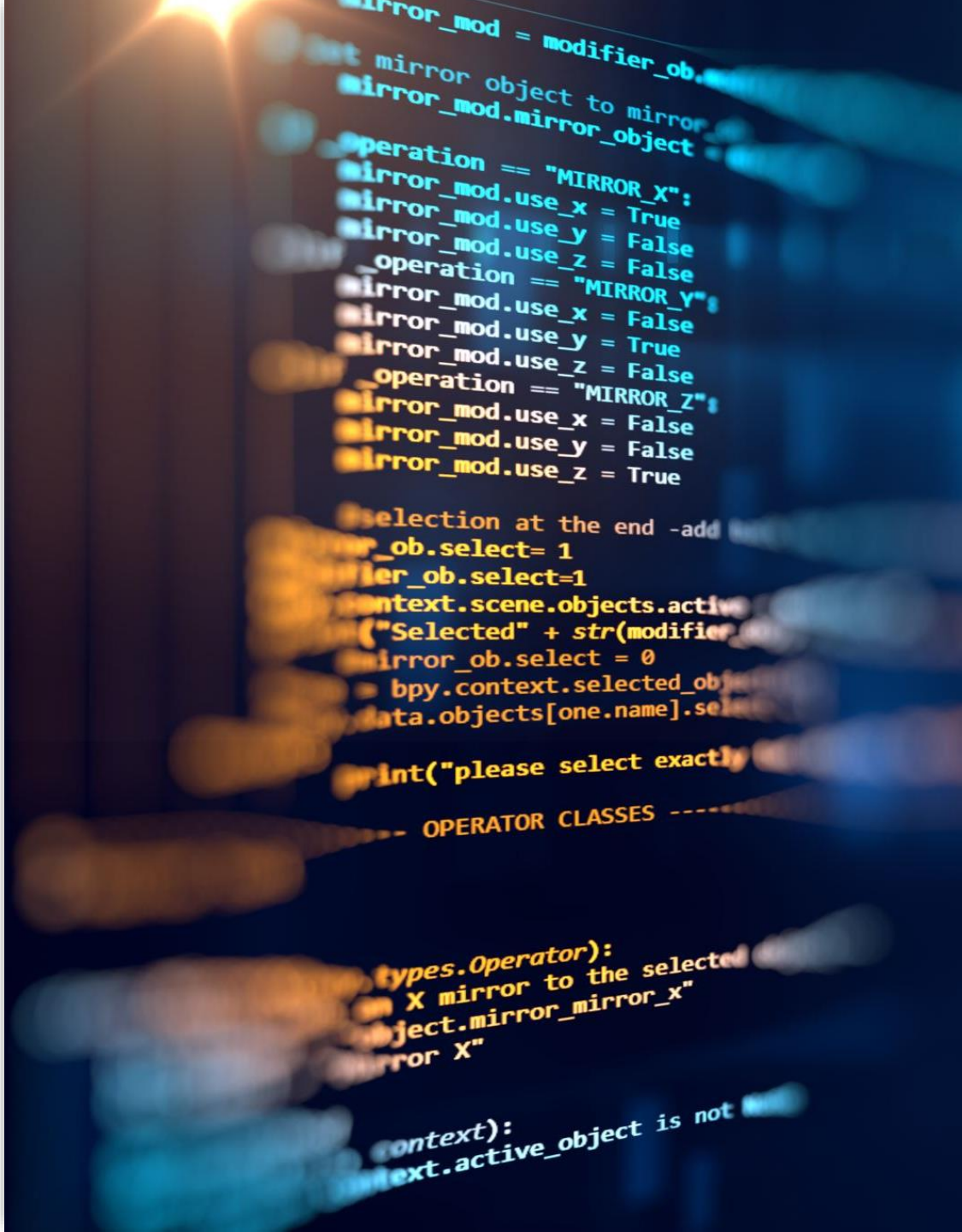
10 exemplos de uso para Web Scraping

1. **Comparação de Preços**: Coleta de preços em sites de e-commerce para comparações e tendências.
2. **Agregação de Notícias**: Reunião de artigos de várias fontes em um único lugar.
3. **Análise de Sentimento**: Coleta de avaliações de redes sociais para insights sobre marcas.
4. **Pesquisa Acadêmica**: Obtenção de dados para estudos em diversas áreas.
5. **Monitoramento de Vagas**: Agregação de oportunidades de emprego de diferentes sites.
6. **Avaliação Imobiliária**: Rastreio de preços de imóveis para análises de mercado.
7. **Dados Meteorológicos**: Coleta de previsões climáticas para estudos.
8. **Identificação de Tendências**: Descoberta de produtos populares para e-commerce.
9. **Análise de Concorrentes**: Monitoramento de mudanças nos sites de concorrentes.
10. **Pesquisa de Mercado**: Coleta de dados setoriais para estratégias de negócios.

Como Funciona o Web Scraping?

1. Enviar uma requisição HTTP para uma página web.
2. Obter e Interpretar a resposta (HTML).
3. Utilizar bibliotecas para extrair informações relevantes.

ROBÓTICA Paula Souza



Ferramentas de Web Scraping

As ferramentas básicas que vamos utilizar neste material:

- Visual Studio Code
- Python 3.11 instalado

Bibliotecas Python que vamos utilizar:

1. requests: Para enviar solicitações HTTP.
2. BeautifulSoup: Para analisar e extrair dados do HTML.

Guia de Instalação

Passo 1: Instalar o Visual Studio Code (VS Code)

1.Acesse o site oficial do VS Code: Download VS Code.

2.Baixe a versão para Windows e execute o instalador.

3.Durante a instalação, marque a opção **"Add to PATH"** para poder usar o comando code no terminal.

4.Complete a instalação seguindo as instruções do instalador.

- **Passo 2: Instalar Python 3.11**

1. Acesse o site oficial do Python: Download Python 3.11.
2. Baixe o instalador do Python 3.11 para Windows.
3. Execute o instalador. Antes de prosseguir, marque a opção **"Add Python 3.11 to PATH"** para facilitar o uso do Python no terminal.
4. Escolha **"Customize installation"** e depois **"Install Now"** para concluir a instalação.

Guia de Instalação

Passo 3: Instalar a Extensão Python no VS Code

1. Abra o VS Code.
2. Vá ao **Marketplace de Extensões** (ícone de quadrado no menu lateral) e procure por "Python".
3. Instale a extensão **Python** da Microsoft para suporte adicional a recursos Python, como depuração, autocompletar e formatação de código.

Passo 4: Verificar a Instalação do Python e pip

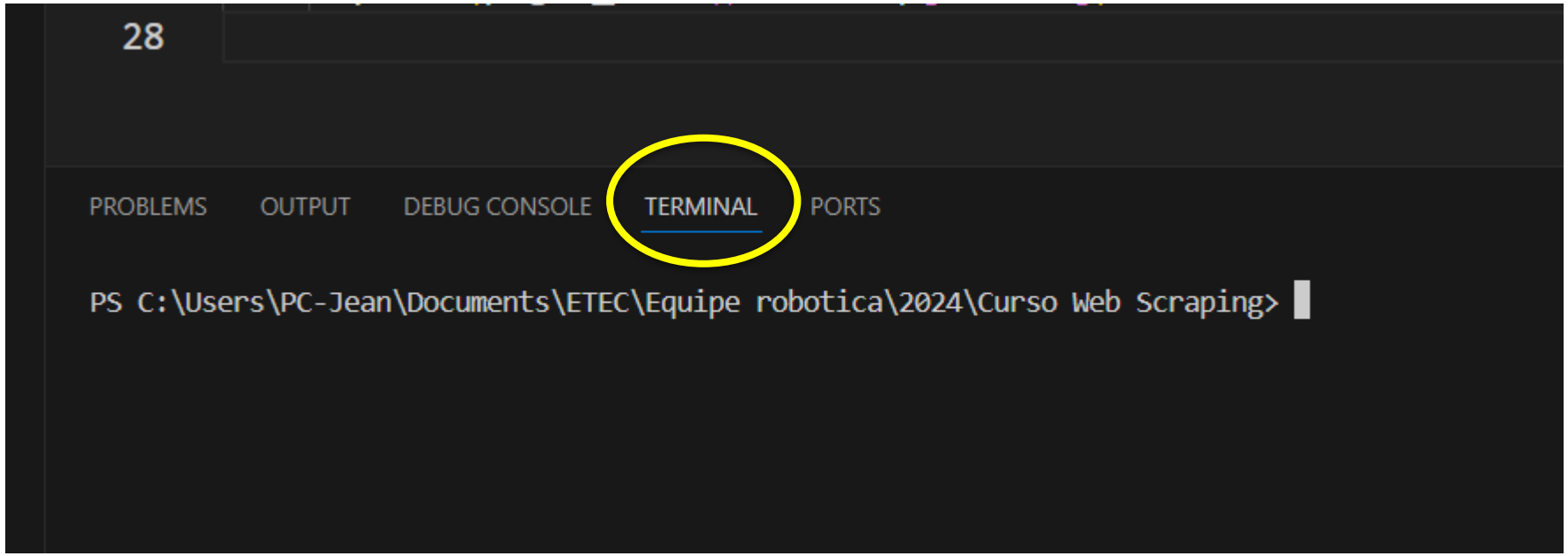
1. Abra o terminal (Prompt de Comando, PowerShell, ou Terminal no VS Code).
2. Digite o comando para verificar a instalação do Python:
3. python --version ou python3 --version
4. Se o comando funcionar, está pronto para uso.

Iniciando um novo projeto

- Criar uma nova Pasta para o projeto
- Abra a Pasta no Visual Studio Code
- Criar um arquivo chamado: main.py

Instalação das Bibliotecas

Abra o Terminal no Visual Studio Code e aplique o comando abaixo:

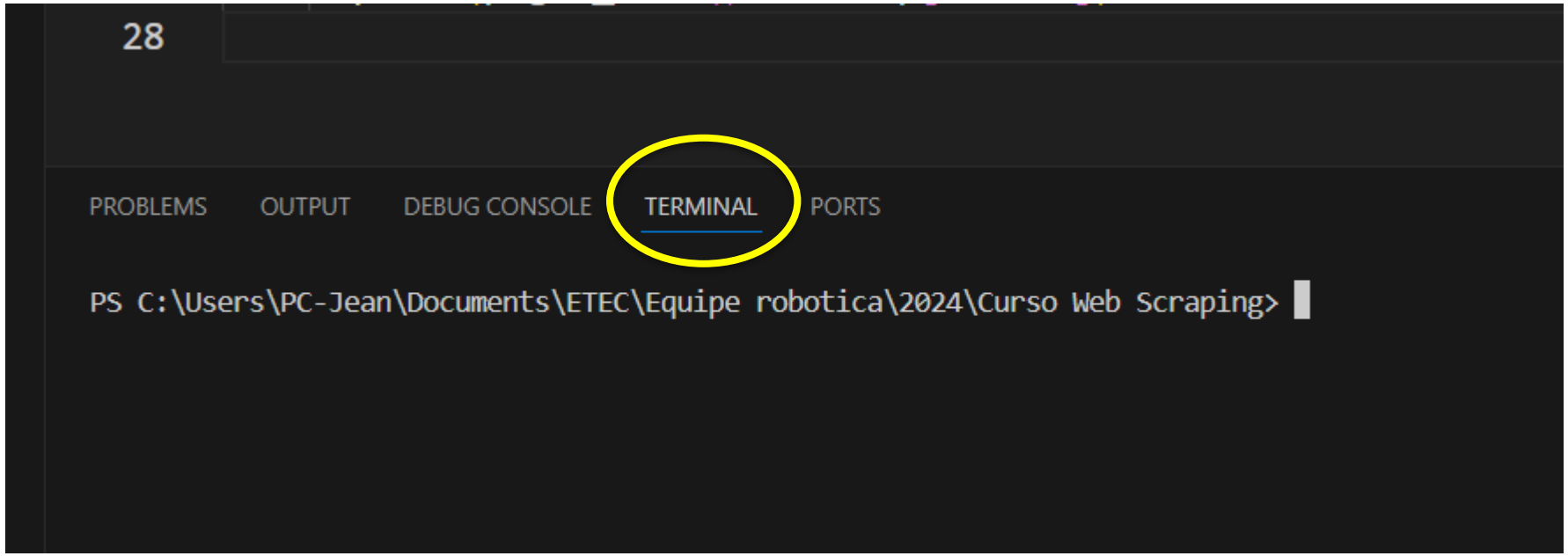


- `pip install requests`

ROBÓTICA Paula Souza

Instalação das Bibliotecas

Abra o Terminal no Visual Studio Code e aplique o comando abaixo:



- `pip install bs4`
- `ou pip install beautifulsoup4`

ROBÓTICA Paula Souza

Exemplo de Requisição HTTP

```
import requests

url = 'https://exemplo.com'
response = requests.get(url)
print(response.text)  # HTML da página
```

Analisando o HTML com BeautifulSoup

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(response.text, 'html.parser')
print(soup.prettify())
# Exibe o HTML de forma legível
```

Exemplo: Obter Todos os Elementos pela Tag

Neste exemplo vamos obter e listar todos os conteúdos das Tags <p> encontradas

```
paragrafos = soup.find_all('p')
for p in paragrafos:
    print(p.get_text())
```

Exemplo: Obter Elemento pelo ID

Neste exemplo vamos obter o conteúdo da Tag especifica através de seu ID

```
elemento = soup.find(id='meu-id')
print(elemento.get_text())
```

Exemplo: Obter Elementos por Classe CSS

Neste exemplo vamos obter o conteúdo de todas Tag que possuam uma classe CSS

```
elementos = soup.find_all(class_='minha-classe')
for el in elementos:
    print(el.get_text())
```

Exemplo: Obter o **href** das Tags <a>

Neste exemplo vamos obter o conteúdo do atributo **href** de todas Tag <a>

```
links = soup.find_all('a')
for link in links:
    print(link.get('href'))
```

python™

Exemplo: Obter **src** das Imagens

Neste exemplo vamos obter o conteúdo do atributo **src** de todas Tag <img>

```
imagens = soup.find_all('img')
for img in imagens:
    print(img.get('src'))
```

Outras dicas

Navegando com Paginação:

- ```
  while next_page is not None:
      # Faz a requisição e extrai dados da próxima página
  ```

Salvando os Dados em CSV

- ```
  import csv

  with open('dados.csv', 'w', newline='') as file:
      writer = csv.writer(file)
      writer.writerow(['Campo1', 'Campo2'])  # Cabeçalhos
      for dado in dados:
          writer.writerow([dado['campo1'], dado['campo2']])
  ```

Boas Práticas no Web Scraping

- Respeito às Regras do Site: Verifique o arquivo robots.txt, ele define as regras que o site estabelece para a indexação de suas páginas e o acesso por bots.
- Ele é uma espécie de "manual de boas práticas" para bots, indicando quais partes do site eles podem ou não acessar.
- Consultá-lo demonstra respeito pelas preferências do site e ajuda a evitar ações não desejadas.

Boas Práticas no Web Scraping

- Evitar Bloqueios e Problemas Legais: Muitos sites monitoram o tráfego gerado por bots.
- Se você acessa áreas que foram explicitamente bloqueadas no robots.txt, isso pode levar a medidas de bloqueio contra o seu endereço IP ou até a problemas legais, dependendo das políticas do site e das leis locais.

Boas Práticas no Web Scraping

- Evitar Carregar Demasiadamente o Servidor: Sites podem restringir áreas específicas ou definir intervalos entre acessos (por meio de Crawl-delay) para evitar sobrecarga em seus servidores.
- Ao seguir essas diretrizes, você ajuda a evitar sobrecarregar o site com requisições excessivas.

Boas Práticas no Web Scraping

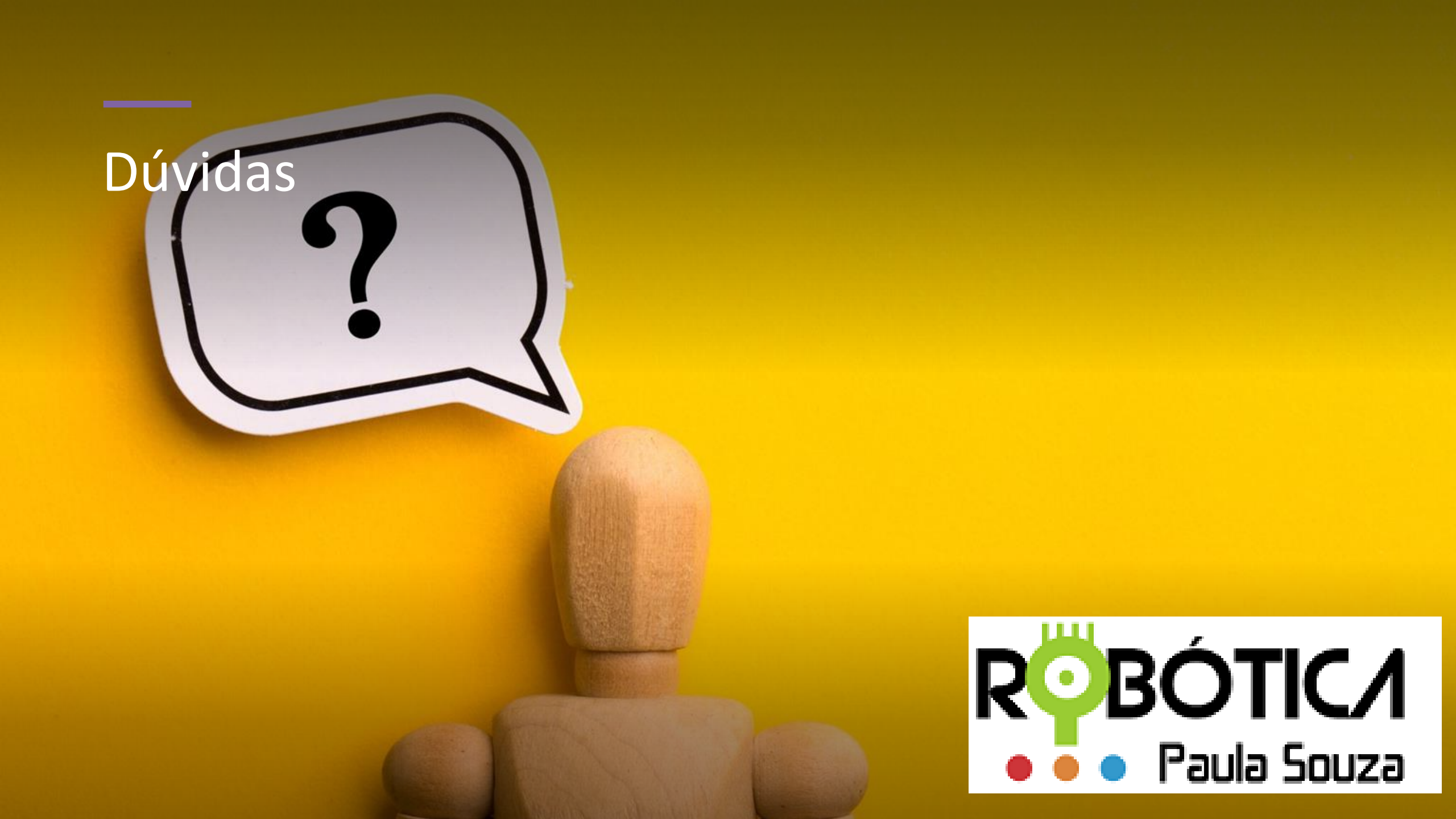
- Conformidade com Padrões de Boa Conduta: Muitos desenvolvedores e empresas que usam web scraping seguem o robots.txt como uma prática de boa conduta.
- Isso ajuda a manter uma relação mais amigável entre desenvolvedores e administradores de sites.

ROBÓTICA Paula Souza

Conclusão

Agora, com uma base sólida sobre Web Scraping, você está preparado para explorar novos sites e coletar dados de forma eficiente e responsável. Lembre-se sempre de respeitar as boas práticas, como verificar o arquivo robots.txt e seguir as políticas do site para evitar sobrecarregar servidores e possíveis problemas legais. O uso de ferramentas como Python e bibliotecas como requests e BeautifulSoup torna esse processo mais acessível e eficiente para diversos objetivos, desde a pesquisa de mercado até a agregação de conteúdo.

Dúvidas



ROBÓTICA
Paula Souza