



PYTHON  
ACADEMY



# WEBSCRAPING COM PYTHON E BEAUTIFULSOUP

Nesse ebook você vai aprender a desenvolver aplicações WebScraping utilizando Python e BeautifulSoup

[PYTHONACADEMY.COM.BR](http://PYTHONACADEMY.COM.BR)

Este ebook foi gerado por



# Crie Ebooks técnicos incríveis em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código!**



Chega de formatar código no Google Docs

Deixe que nossa IA faça o trabalho pesado

 Syntax Highlight

 Adicione Banners Promocionais

 Edite em Markdown em Tempo Real

 Infográficos feitos por IA

**TESTE AGORA** 

 PRIMEIRO CAPÍTULO 100% GRÁTIS

✓ **Atualizado para Python 3.13** (Março 2024)

*BeautifulSoup 4.12+, requests 2.31+, casos práticos de scraping e comparação com outras bibliotecas.*

Salve salve Pythonista!

Hoje, vamos navegar pelo maravilhoso mundo do **Web Scraping** utilizando a popular biblioteca Python, o *BeautifulSoup*.

“Mas o que é Web Scraping?” você deve estar se perguntando. Não se preocupe, a Python Academy está aqui para explicar!

Web Scraping é uma importante ferramenta para coletar dados da web.

O mundo está girando em torno de uma quantidade inacreditável de dados e o Web Scraping é uma bússola, nos ajudando a navegar nesse oceano.

Ao longo deste artigo, vamos explorar tudo sobre Web Scraping com Python e BeautifulSoup.

Vamos visitar temas importantes, como **Crawler**, **Parser**, o uso de BeautifulSoup, seus principais métodos e finalmente, vamos colocar a mão na massa com um mini projeto!

Primeiro, vamos começar do início. Pelos conceitos de **Crawler** e **Parser**!

## Crawler e Parser

Um **Crawler**, também conhecido como *Spider*, é um tipo de script que realiza a varredura na web de maneira metódica e automatizada.

Ele serve para coletar recursos específicos, como, por exemplo, links ou dados específicos de páginas da web.

Suponha que a Página A contenha um link para a Página B e que a Página B possua um link para a Página C.

Através da leitura do código HTML, podemos - de forma automatizada - traçar esse “caminho”, saindo da Página A à Página C, passando pela Página B.

Para isso, basta analisar o código HTML da Página A, em busca de tags `<a>`, como por exemplo:

```
<a href="https://paginab.com.br">Página B</a>
```

E em seguida, realizar a mesma operação na Página B, em busca de outra tag `<a>` que faz referência à Página C, por exemplo:

```
<a href="https://paginac.com.br">Página C</a>
```

Assim como uma aranha “navega” em sua rede, um processo *Spider* (ou Crawler) navega na “Net” (que significa Rede, em Inglês)!

Já o **Parser** é uma ferramenta que nos ajuda a extrair dados de um documento. No mundo do Web Scraping, ele é comumente usado para extrair dados de páginas HTML e XML.

No processo de Parsing, algum programa irá “digerir” os dados “esquisitos” de uma página HTML para uma representação mais simples de ser compreendida e processada, possibilitando a busca e extração de dados!

Esse dois processos (Crawler e Parser) são muito importantes no contexto do Web Scraping.

# BeautifulSoup: Seu Amigo na Coleta de Dados

O *BeautifulSoup* é uma biblioteca Python que facilita a raspagem de informações das páginas da web.

Ele possui um poderoso *parser* que proporciona uma maneira Pythonica de navegar, pesquisar e modificar os dados de um Website.

Para utilizá-lo, utilize o pip e instale o BeautifulSoup **em um ambiente virtual** (se ainda não sabe o que são ambientes virtuais, [clique aqui e aprenda a utilizar o Virtualenv](#)):

```
pip install beautifulsoup4
```

Uma coisa importante a notar aqui é que o BeautifulSoup não baixa as páginas da web para nós. Por isso ele geralmente é utilizado com a biblioteca `requests` para buscar o código HTML de uma página web (instale-o no mesmo ambiente virtual com `pip install requests`).

Sabendo disso, vamos em frente.



# Principais métodos de utilização do BeautifulSoup

Antes de começarmos a usar o BeautifulSoup, devemos dar uma olhada em seus principais métodos:

- `find()` e `find_all()`: Esses dois métodos são usados frequentemente para encontrar tags. O método `find_all()` busca todas as instâncias de uma tag e `find()` busca apenas a primeira instância de uma tag.
- `get()`: Este método é usado para acessar os atributos de uma tag.
- `text`: Esta propriedade é usada para obter o texto de dentro de uma tag.

Agora que conhecemos o básico, vamos utilizar o BeautifulSoup para um exemplo simples de WebScraping:

```
from bs4 import BeautifulSoup
import requests

URL = 'https://www.python.org/'
page = requests.get(URL)

soup = BeautifulSoup(page.content, 'html.parser')

header = soup.find('h1')
print(header)
```

Se você rodar este script, ele retornará o que há dentro da primeira tag '

' que encontrar na página principal do site [python.org](https://python.org), que é o seguinte (no momento em que escrevo este artigo):

```
<h1 class="site-headline">
  <a href="/"></a>
</h1>
```

## Bot de Web Scraping com BeautifulSoup

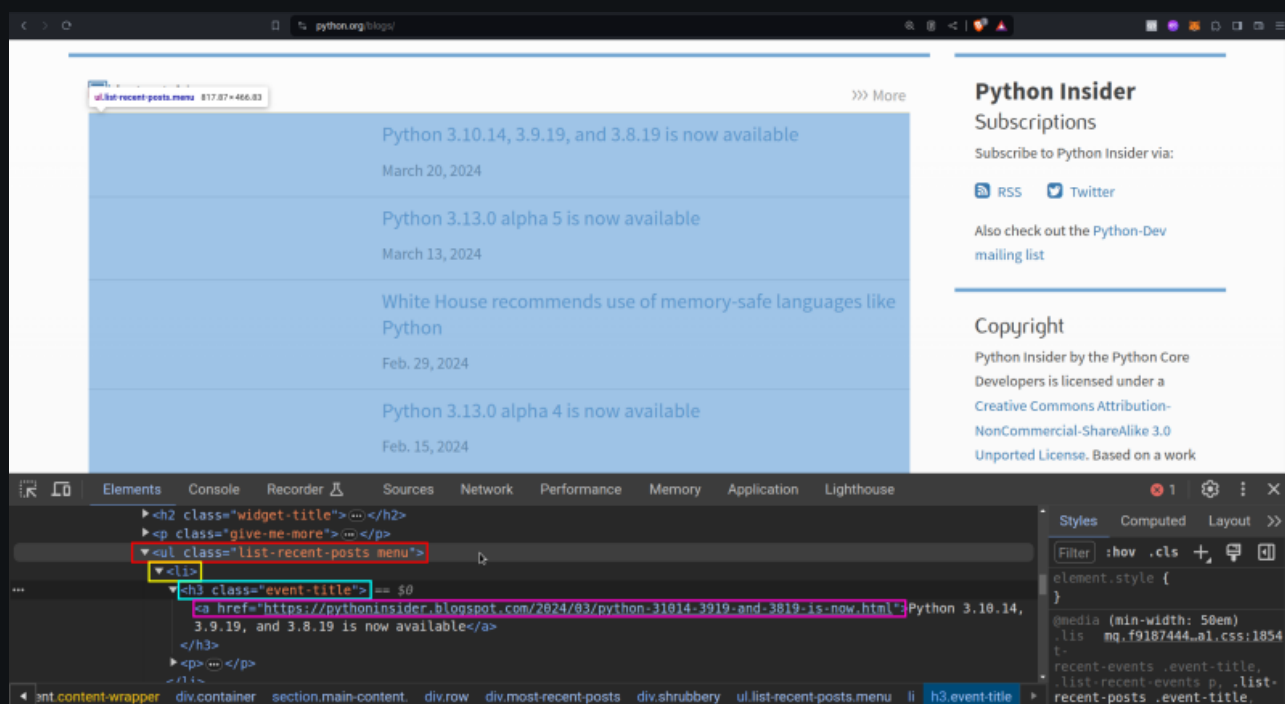
Finalmente, estamos prontos para começar a trabalhar em nosso mini projeto!

Vamos criar um simples bot de raspagem para extrair as últimas notícias do site Python.org.

O primeiro passo para qualquer projeto de Web Scraping é analisar o código HTML da página de onde se quer extrair os dados.

No nosso caso, podemos acessar o website alvo, que no caso é o `www.python.org/blogs` e verificar o código HTML.

Com a ajuda das Ferramentas de Desenvolvedor do Navegador (Chrome ou Firefox, por exemplo), podemos passar o mouse em cima da região onde queremos extrair os dados e analisar a estrutura do código HTML:



Dessa forma, sabemos que o que queremos está: dentro de uma lista `<ul>` com a classe CSS `list-recent-posts`, que contém itens `<li>`, que contém sub-títulos `<h3>` e que contém, por fim, links com a tag `<a>`.

Com isso podemos buscar os dados que queremos da seguinte forma:



```

from bs4 import BeautifulSoup
import requests

URL = 'https://www.python.org/blogs/'
page = requests.get(URL)

soup = BeautifulSoup(page.content, 'html.parser')

# Localizamos a lista <ul> das últimas notícias
news_ul = soup.find('ul', class_='list-recent-posts')

# Encontramos todas as notícias nessa <ul>
latest_news = news_ul.find_all('h3')

# Imprimimos cada notícia, buscando o atributo href de cada link
for news in latest_news:
    link = news.find('a')
    print(f"Artigo: {news.text}\n  Link: {link.attrs['href']}")

```

Este script irá rastrear a página de blogs do site Python.org e imprimirá as últimas notícias em um formato de texto limpo:

```

Artigo: Python 3.10.14, 3.9.19, and 3.8.19 is now available
  Link: https://pythoninsider.blogspot.com/2024/03/python-31014-3919-
and-3819-is-now.html
Artigo: Python 3.13.0 alpha 5 is now available
  Link: https://pythoninsider.blogspot.com/2024/03/python-3130-alp-
ha-5-is-now-available.html
Artigo: White House recommends use of memory-safe languages like Python
  Link: https://pyfound.blogspot.com/2024/02/white-house-recom-
mends-.html
Artigo: Python 3.13.0 alpha 4 is now available
  Link: https://pythoninsider.blogspot.com/2024/02/python-3130-alp-
ha-4-is-now-available.html
Artigo: Software Bill-of-Materials documents are now available for CPy-
thon
  Link: https://pyfound.blogspot.com/2024/02/software-bill-of-materi-
als-now-available-for-cpython.html

```

## Muito maneiro não é mesmo?!

Agora, se quiser aprender mais ainda sobre Web Scraping, eu posso te ajudar!

Em nosso primeiro Projeto da Jornada Python você vai aprender a buscar dados financeiros na internet para criar uma estratégia de investimentos para você!

Além disso, você vai aprender Python do básico ao avançado através de projetos completos e muito exercício.

E não para por aí: você ainda vai aprender a desenvolver aplicações Web utilizando o famoso framework Django!

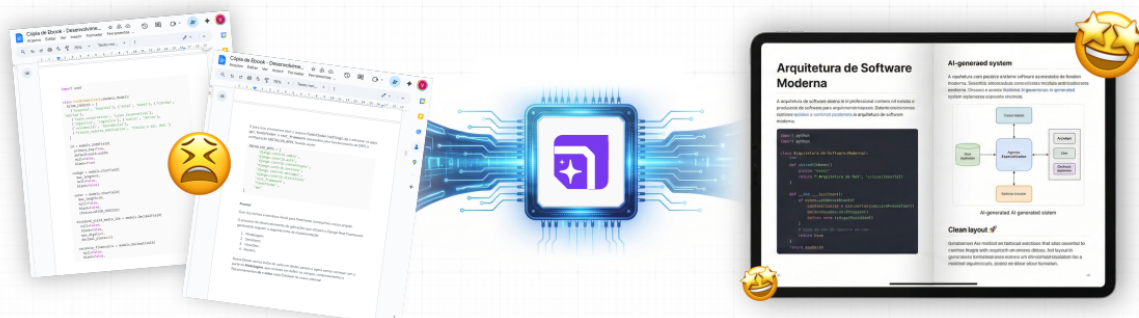
Tudo isso com suporte às suas dúvidas, Comunidade Exclusiva de Alunos, certificado de conclusão, e-books exclusivos e **muito mais!**



*Estou construindo o **DevBook**, uma plataforma que usa IA para criar e-books técnicos — com código formatado e exportação em PDF. Te convido a conhecer!*

## Crie Ebooks técnicos incríveis em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código!**



Chega de formatar código no Google Docs

Deixe que nossa IA faça o trabalho pesado

Syntax Highlight

Adicione Banners Promocionais

Edite em Markdown em Tempo Real

Infográficos feitos por IA

TESTE AGORA! PRIMEIRO CAPÍTULO 100% GRÁTIS

Te vejo na Jornada, caro Pythonista!

## Conclusão

Neste artigo você aprendeu sobre web scraping com Python e BeautifulSoup. Discutimos sobre Crawler e Parser, conhecemos melhor a biblioteca BeautifulSoup e até construímos um simples bot de raspagem.

A combinação de Python e BeautifulSoup torna o web scraping acessível, até mesmo para programadores menos experientes.

Com um pouco de prática, você pode começar a extrair uma grande quantidade de dados da web, que podem ser usados para análise de dados, aprendizado de máquina e uma variedade de outras aplicações.

# BeautifulSoup vs Outras Bibliotecas

Critério	BeautifulSoup	Scrapy	Selenium
<b>Facilidade</b>	✓ Muito fácil	⚠ Médio	✓ Fácil
<b>Performance</b>	⚠ Média	✓ Muito rápida	✗ Lenta
<b>JavaScript</b>	✗ Não suporta	✗ Não suporta	✓ Suporta
<b>Projetos grandes</b>	⚠ Limitado	✓ Ideal	⚠ OK
<b>Curva aprendizado</b>	✓ Baixa	✗ Alta	⚠ Média
<b>Uso ideal</b>	Sites estáticos	Crawling em escala	SPAs/JavaScript

## Quando NÃO Usar BeautifulSoup

### ✗ Site usa JavaScript pesado

Use Selenium ou Playwright em vez disso

### ✗ Precisa scraping em grande escala

Scrapy é 10x mais rápido para projetos grandes

## ❌ Site tem API oficial

Sempre prefira APIs quando disponíveis

## ❌ Site bloqueia scraping agressivamente

Precisará de proxies, headers personalizados e rate limiting

# Casos de Uso Avançados

## 1. Scraping com Headers Personalizados

```
import requests
from bs4 import BeautifulSoup

# Simular navegador real para evitar bloqueios
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36',
    'Accept': 'text/html,application/xhtml+xml',
    'Accept-Language': 'pt-BR,pt;q=0.9',
}

response = requests.get('https://exemplo.com', headers=headers)
soup = BeautifulSoup(response.content, 'html.parser')
```

## 2. Tratamento de Erros Robusto

```
import requests
from bs4 import BeautifulSoup
import time

def scrape_com_retry(url, max_tentativas=3):
    """Scraping com retry automático em caso de falha"""
    for tentativa in range(max_tentativas):
        try:
            response = requests.get(url, timeout=10)
            response.raise_for_status() # Lança exceção se status !=
            200

            soup = BeautifulSoup(response.content, 'html.parser')
            return soup

        except requests.RequestException as e:
            print(f"Tentativa {tentativa + 1} falhou: {e}")
            if tentativa < max_tentativas - 1:
                time.sleep(2 ** tentativa) # Backoff exponencial
            else:
                raise
```



### 3. Extração de Dados Estruturados

```
def extrair_produtos_ecommerce(url):  
    """Extrai dados estruturados de produtos"""  
    response = requests.get(url)  
    soup = BeautifulSoup(response.content, 'html.parser')  
  
    produtos = []  
    cards = soup.find_all('div', class_='product-card')  
  
    for card in cards:  
        produto = {  
            'nome': card.find('h3').text.strip(),  
            'preco': float(card.find('span',  
class_='price').text.replace('R$', '').replace(',', '.')),  
            'avaliacao': float(card.find('span',  
class_='rating').text),  
            'link': card.find('a')['href']  
        }  
        produtos.append(produto)  
  
    return produtos
```

## Boas Práticas

✓ **Use parsers adequados** - `html.parser` - Built-in, sem dependências - `lxml` - Mais rápido, requer instalação - `html5lib` - Mais tolerante a HTML mal-formatado

✓ **Respeite robots.txt**

```
import urllib.robotparser

rp = urllib.robotparser.RobotFileParser()
rp.set_url('https://exemplo.com/robots.txt')
rp.read()

if rp.can_fetch('*', 'https://exemplo.com/pagina'):
    # OK para fazer scraping
    pass
```

## ✓ Implemente rate limiting

```
import time
from datetime import datetime

class RateLimiter:
    def __init__(self, requests_por_segundo=1):
        self.delay = 1.0 / requests_por_segundo
        self.last_request = 0

    def wait(self):
        elapsed = time.time() - self.last_request
        if elapsed < self.delay:
            time.sleep(self.delay - elapsed)
        self.last_request = time.time()

# Uso
limiter = RateLimiter(requests_por_segundo=2)
for url in urls:
    limiter.wait()
    response = requests.get(url)
```

Até a próxima vez, continue programando e se divertindo! 🕷️

# Conclusão

Neste guia sobre **BeautifulSoup**, você aprendeu:

- ✓ **Conceitos fundamentais** - Crawler vs Parser
- ✓ **Instalação e setup** - BeautifulSoup + requests
- ✓ **Métodos principais** - find(), find\_all(), select()
- ✓ **Projeto prático** - Scraping de blog real
- ✓ **Casos avançados** - Headers, retry, dados estruturados
- ✓ **Comparações** - BeautifulSoup vs Scrapy vs Selenium
- ✓ **Boas práticas** - robots.txt, rate limiting, tratamento de erros

**Principais lições:** - BeautifulSoup é **ideal para iniciantes** e sites estáticos - Use **lxml parser** para melhor performance - **Sempre trate erros** e implemente retry - **Respeite rate limits** para evitar bloqueios - Para JavaScript, use **Selenium ou Playwright**

**Próximos passos:** - Explore [Web Scraping com IA \(LangChain\)](#) - Aprenda sobre [APIs vs Scraping](#) - Estude Scrapy para projetos grandes - Pratique com datasets reais

Não se esqueça de conferir!



DevBook

# Crie Ebooks técnicos em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código!**



Chega de formatar código no Google Docs



 Syntax Highlight

 Infográficos feitos por IA

 Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado

 Edite em Markdown em Tempo Real

**TESTE AGORA** 

 PRIMEIRO CAPÍTULO 100% GRÁTIS