

# Scraper de Preços com Python: Automação de Coleta e Exportação de Dados

\*Requisição e leitura do HTML

\*Requisição e leitura do HTML

*\*Desenvolvido por Giovane Ramos – Soluções sob medida com automação em Python*

---

## Introdução

### Sobre o Projeto:

Esse projeto foi criado com o objetivo de demonstrar como é possível utilizar Python para automatizar a coleta de dados da web (Web Scraping), organizando e exportando essas informações de forma prática para análise em Excel ou integração com outros sistemas.

---

## #Tecnologias Utilizadas

- Python 3
  - Biblioteca Requests
  - Biblioteca BeautifulSoup
  - Exportação com módulo CSV
  - IDE: Visual Studio Code
- 

### \* Etapas do Projeto

#### 1. Simulação de URLs

O projeto inicia com URLs simuladas representando páginas de produtos. Isso permite testar a lógica de scraping em ambiente controlado.

#### 2. Coleta de Dados com BeautifulSoup

O script acessa cada URL, interpreta o HTML e extrai informações simuladas como o nome do produto e o preço.

### ✅ 3. Exportação para CSV

Os dados extraídos são organizados em uma planilha .csv, pronta para ser aberta no Excel ou ser consumida por outros sistemas.

### ✅ 4. Execução Real via Terminal

A execução é realizada via terminal, exibindo a mensagem de sucesso:

**" Arquivo precos.csv criado com sucesso."**

### ✅ 5. Planilha gerada.

Resultado final dos dados simulados.

### ✅ 6. Aplicações reais.

**Resolvendo problemas reais.**

---

## #Tecnologias Utilizadas da Lógica do Código

```
src > scraper.py > ...
1 import requests
2 from bs4 import BeautifulSoup
3 import csv
4
5 # URLs simuladas
6 urls = [
7     "https://example.com/produto-a",
8     "https://example.com/produto-b"
9 ]
10
11 headers = {"User-Agent": "Mozilla/5.0"}
12 resultados = []
13
14 for url in urls:
15     r = requests.get(url, headers=headers)
16     soup = BeautifulSoup(r.text, "html.parser")
17
18     nome = soup.title.text.strip() if soup.title else "Produto Desconhecido"
19     preco = "R$ 99,90" # Preço simulado
20
21     resultados.append({"Produto": nome, "Preço": preco, "URL": url})
22
23 with open("precos.csv", "w", newline="", encoding="utf-8") as f:
24     writer = csv.DictWriter(f, fieldnames=["Produto", "Preço", "URL"])
25     writer.writeheader()
26     writer.writerows(resultados)
27
28 print("✓ Arquivo precos.csv criado com sucesso.")
29
```

Essas são as ferramentas que o código usa:

- `requests` para acessar páginas web
- `BeautifulSoup` para ler e interpretar o HTML
- `csv` para exportar os dados coletados em planilha

## 1. `Urls` , para acessar páginas web,

```
5 # URLs simuladas
6 urls = [
7     "https://example.com/produto-a",
8     "https://example.com/produto-b"
9 ]
```

# 7/8 Esses links simulam páginas de produtos. O script irá "visitar" cada um deles.

## 2. `BeautifulSoup` para ler e interpretar o HTML

Requisição e leitura.

```

14 for url in urls:
15     r = requests.get(url, headers=headers)
16     soup = BeautifulSoup(r.text, "html.parser")

```

#15 Faz a requisição com cabeçalho simulado #16 Interpreta o HTML da página. O script acessa o link e lê todo o código da página, como se fosse um navegador.

### 3. **csv** Para exportar os dados coletados em planilha

```

23 with open("precos.csv", "w", newline="", encoding="utf-8") as f:
24     writer = csv.DictWriter(f, fieldnames=["Produto", "Preço", "URL"])

```

#23/24 Cria o arquivo .csv, escreve o cabeçalho e adiciona as linhas com os dados coletados.

Coleta do **<title>**, nome do produto.

```

18 nome = soup.title.text.strip() if soup.title else "Produto Desconhecido"

```

#18 Aqui o código verifica se o HTML da página contém um título ( **<title>** ). Se sim, ele remove espaços extras e usa esse texto como nome do produto. Caso não tenha, atribui "Produto Desconhecido" — o que evita erros.

### Preço simulado

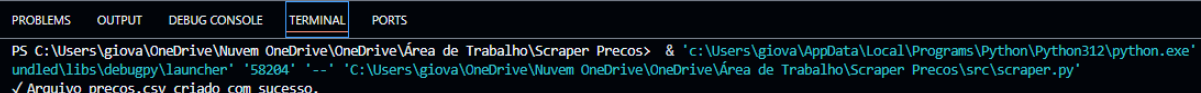
```

19 preco = "R$ 99,90" # Preço simulado

```

#19 Neste exemplo, o preço é fixo, pois não estamos usando um site real ainda. Em projetos reais, esse valor pode ser extraído dinamicamente com base no HTML.


## 4. Mensagem final



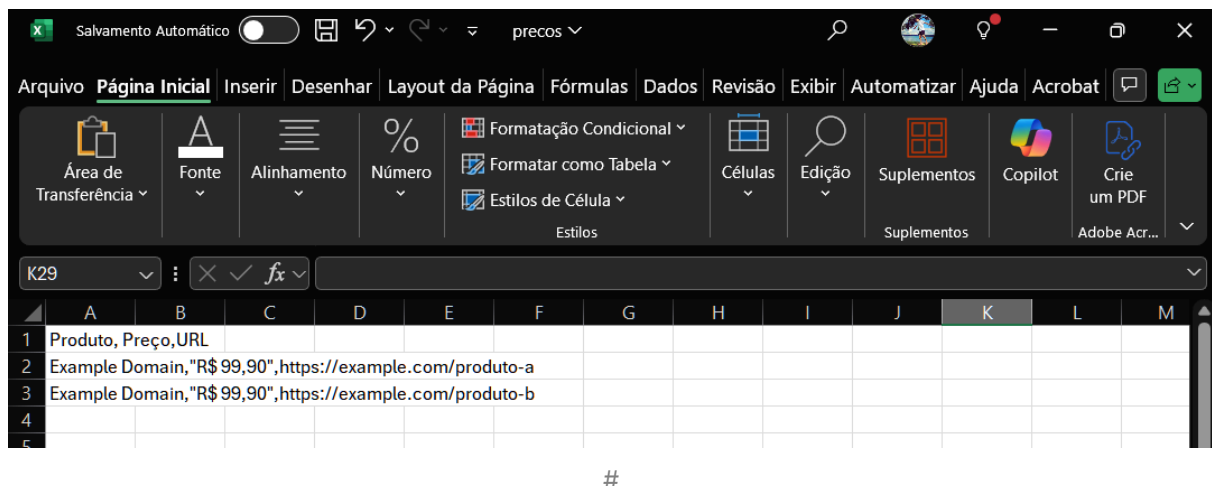
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\giova\OneDrive\Nuvem OneDrive\Área de Trabalho\Scraper Precos> & 'c:\Users\giova\AppData\Local\Programs\Python\Python312\python.exe'
undled\libs\debugpy\launcher' '58204' '-' 'C:\Users\giova\OneDrive\Nuvem OneDrive\Área de Trabalho\Scraper Precos\src\scraper.py'
✓ Arquivo precos.csv criado com sucesso.

```

# Terminal  Essa mensagem no terminal confirma que todo o processo foi executado com sucesso: requisição, scraping e exportação dos dados.

## 5. Resultado final da planilha gerada no Excel com todos os dados simulados.



O arquivo `precos.csv` foi criado automaticamente com os dados coletados.

Ele pode ser aberto no Excel, Power BI, Google Sheets ou integrado com sistemas externos. Isso mostra como esse projeto pode ter valor prático mesmo sendo simples.

## 6. Quais as Aplicações reais do projeto?

- **Monitoramento de preços em e-commerces:** para acompanhar a concorrência ou manter preços dinâmicos.
- **Extração de dados para análise de mercado:** excelente para estratégias comerciais, comparações e tendências.
- **Base para integrações com ERPs e dashboards Power BI:** gera dados padronizados, prontos para automações.

## Conclusão

---


Este projeto representa um exemplo básico, porém funcional, de automação com Python para tarefas de Web **Scraping**. É ideal para empresas que desejam extrair dados da web ou organizar informações de forma automatizada e eficiente.


---

### Sobre o Desenvolvedor

Giovane C Ramos – Desenvolvedor focado em automações com IA + Python.  
Profissional em transição de carreira, entregando soluções com disciplina e visão estratégica.

---

 [giiotec@Outlook.com](mailto:giiotec@Outlook.com)

 [GitHub:github.com/giiotec](https://github.com/giiotec)



Workana:<https://www.workana.com/freelancer/bd6ed22337fd58eb1bf7ebbcbf11639c>