



PYTHON
ACADEMY



PYTHON E LANGCHAIN: WEB SCRAPING COM IA

Nesse ebook, você vai desvendar como usar Langchain e Inteligência Artificial para extrair conteúdos de páginas web

PYTHONACADEMY.COM.BR

Gere ebooks como este com



em <https://ebookr.ai>

Crie ebooks profissionais incríveis em minutos com IA



Chega de formatar texto no Google Docs, Word ou ferramentas que só te fazem perder tempo...

E deixe que nossa IA faça o trabalho pesado!



Capas gerados por IA



Infográficos feitos por IA



Edite em Markdown em Tempo Real



Adicione Banners Promocionais

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS

✓ **Atualizado para LangChain 0.1+ e Playwright 1.40+** (Março 2025) Web scraping com IA usando LangChain, Playwright, extração estruturada com Pydantic e casos práticos reais.

Salve salve Pythonista 🙌

No mundo digital de hoje, o **Web Scraping** é uma técnica essencial para extrair informações de sites.

Com a evolução da inteligência artificial, surge a possibilidade de incorporar agentes inteligentes através de Frameworks LLM, como o **Langchain**, e automação de navegadores, como o **Playwright**, para enriquecer esse processo.

Neste artigo, exploraremos como combinar essas ferramentas para realizar web scraping de maneira eficiente e inteligente.

Vamos entender o que são o Langchain e o Playwright, como configurá-los e implementá-los em um projeto de web scraping.

Além disso, utilizaremos outras bibliotecas úteis e o ChatGPT para analisar os dados extraídos.

Vamos lá!

O que é Langchain?

O **Langchain** é uma biblioteca que simplifica a criação de pipelines de processamento de linguagem natural sofisticados.

Ela permite que desenvolvedores combinem diferentes modelos e técnicas de forma modular.

Utilizando o Langchain, podemos facilmente misturar e combinar modelos de IA para obter uma análise aprimorada do texto extraído durante o web scraping.

Se quiser saber mais sobre o LangChain, [clique aqui para ler um artigo completo com mais detalhes sobre essa formidável ferramenta](#).

O que é Playwright?

Playwright é uma biblioteca para automação de navegadores, semelhante ao Selenium, mas com desempenho melhorado e suporte a multiplataforma.

Ele permite controlar navegadores para extrair dados dinâmicos de sites de forma eficiente, essencial quando lidamos com páginas que carregam dados via JavaScript.

O langchain utiliza o Playwright para carregar páginas dinâmicas e extrair o conteúdo HTML para análise.

Bibliotecas Necessárias

Para implementar nosso projeto de Web Scraping com IA, precisamos instalar as seguintes bibliotecas:

```
pip install playwright \
beautifulsoup4 \
python-decouple \
langchain \
langchain-community \
langchain-openai
```

Explicação das Bibliotecas

- **playwright**: Automação de navegador para carregamento de páginas dinâmicas.
- **beautifulsoup4**: Extração de dados do HTML.
- **python-decouple**: Carregar variáveis de ambiente de arquivos `.env`.
- **langchain**: Construção de pipelines de linguagem natural.
- **langchain-community**: Funcionalidades adicionais para o Langchain.
- **langchain-openai**: Integração com a API da OpenAI.

Implementação do Web Scraping com IA

Vamos implementar o processo de web scraping com as etapas detalhadas a seguir.

Configuração Inicial

Primeiro, vamos configurar nossa chave de API e o modelo de linguagem natural que usaremos:

Crie um arquivo `.env` na raiz do projeto com a chave da API da OpenAI:

```
from decouple import config
import os

# Carrega a chave da API da OpenAI do arquivo .env usando decouple
os.environ['OPENAI_API_KEY'] = config('OPENAI_API_KEY')
```

Definição do Modelo

Definimos o modelo para estruturar os dados do artigo, utilizando o Pydantic.

Estruturamos o modelo do artigo com campos como título, descrição, data, autor e link:

```
from pydantic import Field, BaseModel

class Article(BaseModel):
    """
    Representa um artigo com informações estruturadas.
    """
    headline: str = Field(description="O título do artigo")
    description: str = Field(description="Uma breve descrição do artigo")
    date: str = Field(description="A data em que o artigo foi publicado")
    author: str = Field(description="O autor do artigo")
    link: str = Field(description="O link para o artigo")
```

Carregamento de Dados

Usamos o **AsyncChromiumLoader** do Langchain para carregar o conteúdo HTML das URLs. Ele carrega o conteúdo HTML das URLs especificadas:

```
from langchain_community.document_loaders import AsyncChromiumLoader

# Lista de URLs que serão carregadas
urls = ["https://pythonacademy.com.br/blog/"]

loader = AsyncChromiumLoader(urls)
docs = loader.load()
```

Extração com BeautifulSoup

Com o **BeautifulSoupTransformer**, extraímos as tags desejadas:

Dentro da função `transform_documents`, passamos a lista de documentos carregados e as tags que queremos extrair (como “article”) para obter as tags que você precisa inspecionar o HTML da página e identificar as tags que contêm o conteúdo desejado.:

```
from langchain_community.document_transformers import BeautifulSoupTransformer

# Utiliza BeautifulSoupTransformer para extrair apenas as tags
# desejadas ("article")
bs_transformed = BeautifulSoupTransformer()
docs_transformed = bs_transformed.transform_documents(
    documents=docs,
    tags_to_extract=["article"]
)
```

Tokenização de Texto

Usamos o **RecursiveCharacterTextSplitter** para tokenizar o texto entregue para análise:

```

from langchain_text_splitters import RecursiveCharacterTextSplitter

# Divide o conteúdo extraído em pedaços usando tokenização
splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
    chunk_size=2000,
    chunk_overlap=0
)
splits = splitter.split_documents(documents=docs_transformed)

```

Análise de Conteúdo

Finalmente, invocamos o modelo estruturado de linguagem natural para estruturar os dados:

```

from langchain_openai import ChatOpenAI

# Cria uma instância do modelo ChatOpenAI configurado com GPT-4o-mini
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

# Configura o modelo para retornar respostas estruturadas no formato da
# classe Article
structured_llm = llm.with_structured_output(Article)

# Extrai e estrutura as informações usando o modelo configurado para
# cada pedaço
extracted_content = [structured_llm.invoke(split.page_content) for
    split in splits]

# Converte o conteúdo estruturado em uma lista de dicionários
extracted_content_dict = [article.model_dump() for article in
    extracted_content]

# Exibe o resultado
print(extracted_content_dict)

```


Resultado da Extração de Dados

O código acima extrai e estrutura dados das páginas especificadas retornando uma lista de artigos:

```
[
  {'headline': 'Padrões de Projeto em Python (Design Patterns)',
    'description': 'Neste artigo, você aprenderá sobre Padrões de Projeto em Python (Design Patterns) e sua importância.',
    'date': '26/02/2025',
    'author': 'Vinícius Ramos',
    'link': 'https://pythonacademy.com.br/blog/padroles-de-projeto-em-python'},

  {'headline': 'O que é o Django Rest Framework (DRF)?',
    'description': 'Entenda o que é o Django Rest Framework (DRF) e como ele pode facilitar o desenvolvimento de APIs REST em Django.',
    'date': '27/08/2024',
    'author': 'Vinícius Ramos',
    'link': 'https://pythonacademy.com.br/blog/o-que-e-o-django-rest-framework'}
]
```

Este resultado revela como o uso de IA pode facilitar e estruturar processos de extração de dados.

Código Completo do Web Scraping com IA

Aqui está o código completo para referência:

```

from decouple import config
from langchain_community.document_loaders import AsyncChromiumLoader
from langchain_community.document_transformers import BeautifulSoupTransformer

from langchain_openai import ChatOpenAI
from langchain_text_splitters import RecursiveCharacterTextSplitter
from pydantic import Field, BaseModel
import os

os.environ['OPENAI_API_KEY'] = config('OPENAI_API_KEY')

llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

class Article(BaseModel):
    headline: str = Field(description="O título do artigo")
    description: str = Field(description="Uma breve descrição do artigo")
    date: str = Field(description="A data em que o artigo foi publicado")
    author: str = Field(description="O autor do artigo")
    link: str = Field(description="O link para o artigo")

structured_llm = llm.with_structured_output(Article)

urls = ["https://pythonacademy.com.br/blog/"]
tags_to_extract = ["article"]

loader = AsyncChromiumLoader(urls)
docs = loader.load()

bs_transformed = BeautifulSoupTransformer()
docs_transformed = bs_transformed.transform_documents(
    documents=docs,
    tags_to_extract=tags_to_extract
)

splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
    chunk_size=2000,
    chunk_overlap=0
)

splits = splitter.split_documents(documents=docs_transformed)

```

```
extracted_content = [structured_llm.invoke(split.page_content) for
split in splits]

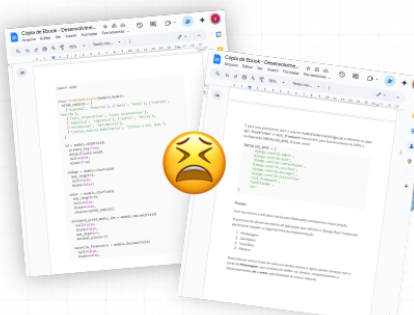
extracted_content_dict = [article.model_dump() for article in
extracted_content]

print(extracted_content_dict)
```

Falando em LangChain e IA: Estou usando exatamente essas tecnologias para construir o **Ebookr.ai** — uma plataforma que gera ebooks profissionais sobre qualquer tema automaticamente. O mesmo poder do LangChain que você está aprendendo aqui, aplicado para criar conteúdo de alta qualidade. Confira!



Crie Ebooks profissionais incríveis em minutos com IA



Chega de formatar texto no Google Docs, Word ou ferramentas que só te fazem perder tempo...





... e deixe que nossa IA faça o trabalho pesado!

TESTE AGORA! PRIMEIRO CAPÍTULO 100% GRÁTIS

Capas gerados por IA

Adicione Banners Promocionais

Edite em Markdown em Tempo Real

Infográficos feitos por IA

Conclusão

Neste artigo, mergulhamos na combinação poderosa do **Langchain** e do **Playwright** para realizar web scraping avançado com IA.

Estudamos como configurar e implementar estas ferramentas, além de outras bibliotecas essenciais, para extrair dados de sites de forma eficiente.

Com a capacidade de estruturar informações de maneira precisa, esse método transforma a maneira como abordamos a extração de dados na web.

Experimente estas técnicas no seu próximo projeto e veja os benefícios dessa abordagem inovadora.

LangChain + Playwright vs Outras Abordagens

Critério	LangChain+Playwright	BeautifulSoup	Scrapy	Selenium
JavaScript	✓ Executa	✗ Não	✗ Não	✓ Executa
Extração IA	✓ Built-in	✗ Manual	✗ Manual	✗ Manual
Estruturação	✓ Pydantic	✗ Manual	✗ Manual	✗ Manual
Performance	⚠ Média	✓ Rápida	✓ Muito rápida	✗ Lenta
Custo	✗ APIs (\$\$)	✓ Grátis	✓ Grátis	✓ Grátis
Precisão	✓ Alta (IA)	⚠ Média	⚠ Média	⚠ Média
HTML dinâmico	✓ Simples	✗ Não	✗ Não	✓ Simples
Uso ideal	SPAs + IA	Estático	Escala	SPAs básico

Quando Usar LangChain + Playwright

- ✓ **Sites JavaScript pesados (SPAs)** Playwright renderiza JavaScript completo
- ✓ **Precisa extração inteligente** IA entende contexto e estrutura dados
- ✓ **HTML mal estruturado/inconsistente** IA adapta-se a variações na estrutura
- ✓ **Dados não estruturados em texto** Exemplo: extrair preços de parágrafos
- ✓ **Prototipar rapidamente** Menos código que parsers tradicionais

Quando NÃO Usar

- ✗ **Site tem API oficial** API é sempre melhor e mais confiável
- ✗ **Orçamento limitado** Custos de API OpenAI podem ser altos
- ✗ **Precisa de escala massiva** Scrapy é muito mais rápido/barato
- ✗ **HTML bem estruturado e estático** BeautifulSoup é suficiente e mais barato
- ✗ **Performance crítica** Playwright + IA adiciona latency

Casos de Uso Avançados

1. Monitoramento de Preços com IA

```
from langchain_community.document_loaders import PlaywrightURLLoader
from langchain.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field
from typing import List

class Produto(BaseModel):
    nome: str
    preco_atual: float
    preco_original: float = None
    desconto_percentual: float = None
    disponivel: bool

# Scraping com extração estruturada
loader = PlaywrightURLLoader(["https://ecommerce.com/produto"])
docs = loader.load()

parser = PydanticOutputParser(pydantic_object=Produto)
produto = parser.parse(docs[0].page_content)

print(f"Preço atual: R$ {produto.preco_atual}")
if produto.desconto_percentual:
    print(f"Desconto: {produto.desconto_percentual}%")
```

2. Extração de Dados de Tabelas Complexas

```
from pydantic import BaseModel
from typing import List

class LinhaTabela(BaseModel):
    empresa: str
    receita: float
    lucro: float
    margem: float

class TabelaFinanceira(BaseModel):
    titulo: str
    linhas: List[LinhaTabela]
    total_empresas: int

# IA extrai automaticamente estrutura da tabela
loader = PlaywrightURLLoader(["https://site.com/tabela-financeira"])
docs = loader.load()

parser = PydanticOutputParser(pydantic_object=TabelaFinanceira)
tabela = parser.parse(docs[0].page_content)

for linha in tabela.linhas:
    print(f"{linha.empresa}: {linha.margem}% de margem")
```

3. Scraping com Interação (Login, Scroll)

```
from playwright.sync_api import sync_playwright

def scrape_com_interacao(url, username, password):
    with sync_playwright() as p:
        browser = p.chromium.launch(headless=False)
        page = browser.new_page()

        # Login
        page.goto(url)
        page.fill('input[name="username"]', username)
        page.fill('input[name="password"]', password)
        page.click('button[type="submit"]')

        # Aguardar carregamento
        page.wait_for_selector('.dashboard')

        # Scroll infinito
        previous_height = 0
        while True:
            page.evaluate('window.scrollTo(0, document.body.scrollHeight)')
            page.wait_for_timeout(2000)

            new_height = page.evaluate('document.body.scrollHeight')
            if new_height == previous_height:
                break
            previous_height = new_height

        # Extrair conteúdo
        content = page.content()
        browser.close()

    return content
```

Otimização de Custos

1. Cache de Conteúdo

```
import hashlib
import json
from pathlib import Path

def scrape_com_cache(url):
    # Hash da URL como chave de cache
    cache_key = hashlib.md5(url.encode()).hexdigest()
    cache_file = Path(f"cache/{cache_key}.json")

    # Verificar cache
    if cache_file.exists():
        with open(cache_file) as f:
            return json.load(f)

    # Scraping real (custa API)
    loader = PlaywrightURLLoader([url])
    docs = loader.load()
    result = extract_with_ai(docs[0].page_content)

    # Salvar em cache
    cache_file.parent.mkdir(exist_ok=True)
    with open(cache_file, 'w') as f:
        json.dump(result, f)

    return result
```

2. Processar em Batch

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

# Processar múltiplas páginas juntas (economiza tokens)
urls = ["url1", "url2", "url3"]
loader = PlaywrightURLLoader(urls)
docs = loader.load()

# Combinar antes de processar
combined_content = "\n\n".join([doc.page_content for doc in docs])

# Processar tudo de uma vez
result = extract_all_with_single_api_call(combined_content)
```

Conclusão

Neste guia sobre **Web Scraping com IA**, você aprendeu:

✓ **Combinação poderosa** - LangChain + Playwright + IA ✓ **Extração estruturada** - Pydantic para dados tipados ✓ **JavaScript** - Playwright renderiza SPAs ✓ **Comparações** - vs BeautifulSoup, Scrapy, Selenium ✓ **Quando usar** - SPAs, HTML inconsistente, extração inteligente ✓ **Otimização** - Cache e batch para reduzir custos ✓ **Casos avançados** - Login, scroll, tabelas complexas

Principais lições:

- **IA + scraping** = extração inteligente e adaptativa
- **Playwright** executa JavaScript, BeautifulSoup não
- **Pydantic** estrutura dados automaticamente
- **Custos de API** podem ser altos - use cache
- Para sites simples, **BeautifulSoup é melhor**

Próximos passos:

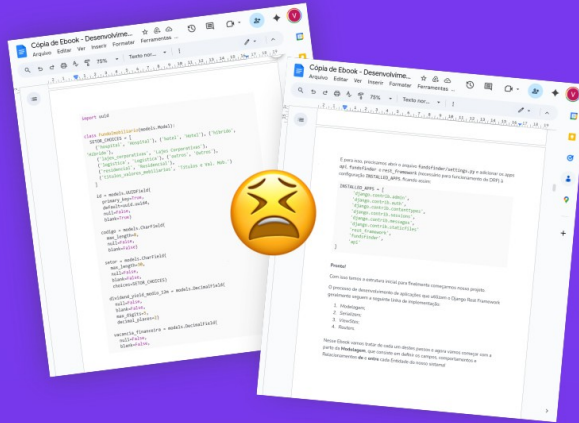
- Aprenda [LangChain básico](#)
- Explore [Prompts no LangChain](#)
- Estude sobre [embeddings e RAG](#)
- Pratique com Playwright Codegen

Não se esqueça de conferir!



Ebookr

Crie Ebooks profissionais em minutos com IA



Chega de formatar código no Google Docs ou Word



Capas gerados por IA



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