Raspagem de dados com Python: Um guia para iniciantes

Paulo Henrique Santos Gonçalves

Biólogo com doutorado em Diversidade Biológica e Conservação nos Trópicos

Bolsista de pós-doutorado pela Universidade Federal de Pernambuco

Outubro de 2025

Informações de financiamento

Este material foi desenvolvido com apoio de bolsa de pós-doutorado concedida pela FACEPE (Fundação de Amparo à Ciência e Tecnologia do Estado de Pernambuco).

Apresentação

Este e-book nasceu como guia para auxiliar estudantes interessados em começar a aprender técnicas de web scraping (ou raspagem de dados, em português). Como eu sou biólogo de formação, experienciei dificuldades ao tentar dar os primeiros passos no estudo dessas técnicas. A maior parte da literatura atualizada sobre o tema é em inglês e possui linguagem muito técnica para um principiante. Por tratar-se de um guia inicial, eu trarei aqui as técnicas mais básicas de raspagem de dados, as quais usam, principalmente, a biblioteca Beautiful Soup.

Se para você este assunto é completamente novo, web scraping consiste no uso de técnicas de programação que visam a construir códigos capazes de visitar uma ou muitas páginas da internet com a intenção de coletar e analisar dados de maneira automática.

Imagine, por exemplo, que uma empresa lança um novo produto e quer monitorar as resenhas dos clientes em um determinado site para saber se o produto está sendo bem avaliado. Em vez de visitar esse site manualmente todos os dias, é possível elaborar um código de raspagem de dados que monitore as resenhas e analise os principais sentimentos expressos pelos usuários. Outro exemplo que vi recentemente numa rede social diz respeito à possibilidade de monitorar preços de produtos em supermercado ao longo do tempo.

Raspagem de dados também possui muita aplicabilidade para pesquisas científicas, visto que há um número crescente de empresas e instituições públicas que disponibilizam informações na internet, as quais podem ser usadas para responder a perguntas científicas. Muitas vezes, entretanto, essas informações não são facilmente disponíveis, como em planilhas de Excel. Nesses casos, é possível que o pesquisador precise aprender um pouco sobre raspagem de dados para coletar as informações desejadas.

Raspagem de dados pode ser feito usando diferentes linguagens de programação. Neste e-book, eu usarei a linguagem Python, visto que esta é considerada uma das linguagens mais simples e versátil para esta técnica. Neste momento, eu vou supor que você já tem o Python instalado e que você sabe fazer funcionalidades básicas nesta linguagem. Portanto, este não é um guia introdutório ao uso do Python. Outro ponto a ser ressaltando é que raspagem de dados é uma técnica muito dinâmica porque a estrutura

dos sites pode mudar ao longo do tempo. Assim, não desanime se algum dos exemplos que eu trouxer aqui não funcionar mais no futuro. O objetivo é compreender os fundamentos básicos. Nas próximas páginas, eu apresento alguns conceitos fundamentais que é preciso entender antes de fazer uma primeira raspagem de dados.

Sumário

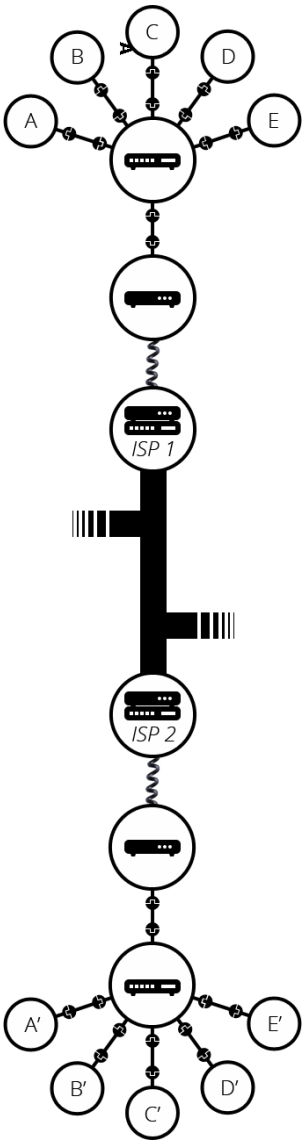
Capítulo 1: Conceitos fundamentais

Introdução

De maneira bastante simplificada, a internet pode ser compreendida como uma gigantesca rede de computadores capazes de comunicarem-se entre si. A maneira mais simples pela qual podemos conectar dois computadores é fisicamente (por meio de um cabo de rede) ou sem fio (usando sistemas WiFi ou Bluetooth). Porém, é impossível conectar todos os computadores dessa forma. Assim, para resolver esse problema, cada computador está conectado a um pequeno computador especial chamado de roteador. A função do roteador é fazer com que a mensagem de um computador seja entregue ao computador destinatário correto. Uma vez que um roteador é um computador como qualquer outro, a conexão entre diferentes roteadores permite que um número infinitamente grande de computadores possa conectar-se (Mitchell, 2024).

Nós não podemos adicionar cabos aos nossos computadores para conectar a outros computadores distantes, como os de nosso trabalho ou de nossos amigos. Para resolver esse problema, usamos uma estrutura que já está presente na maior parte do mundo, que é a estrutura do telefone que temos em casa. Nossa rede de computadores é conectada à rede de telefone por meio de um equipamento denominado modem. O modem transforma a informação da nossa rede de computadores em uma informação gerenciável pela rede telefônica e vice-versa (Mitchell, 2024).

Finalmente, para enviar mensagens de nossa rede para outra rede qualquer precisamos nos conectar a um Provedor de Serviço de Internet (ISP, em inglês). Um ISP é uma empresa que gerencia alguns roteadores especiais que são conectados e podem também acessar roteadores de outros ISPs. Assim, a mensagem da nossa rede é transportada para as redes do ISP e, então, para a rede de destino. Dessa forma, a internet é uma infraestrutura técnica que permite conectar bilhões de computadores (Mitchell, 2024).



A rede de internet pode ser compreendida como um sistema que conecta bilhões de computadores em todo o mundo e permite que estes comuniquem-se por meio de servidores, rede de telefonia e Provedores de Serviço de Internet (ISP)

Figura 1. Representação esquemática do funcionamento da internet. Cada letra representa um computador com cabos conectando-os a um roteador. Os roteadores conectam-se entre si por meio da rede telefônica e Provedores de Serviço de Internet. Figura do site MDN Web Docs – Mozilla (MDN, 2025).

A compreensão do conceito de servidor web demanda que saibamos, primeiramente diferenciar hardware e software. Hardware refere-se aos componentes físicos de um computador ou dispositivo eletrônico, tais como a CPU (processador), memória RAM, monitor, teclado ou mouse. O software refere-se ao conjunto de programas, sistemas operacionais e aplicativos que orientam o hardware a realizar tarefas específicas, tais como Sistemas operacionais (Windows, Linux, macOS) ou aplicativos (Word, Excel, navegadores).

Dentro do conceito de hardware, o servidor web pode ser compreendido como um computador que armazena arquivos que compões os sites (incluindo o aspecto estrutural,

visual e funcional do site) e os entrega para o computador do usuário final. Dentro do conceito de software, o servidor web é um programa responsável por gerenciar e responder às solicitações de usuários que querem acessar conteúdos hospedados em sites. O principal componente do servidor web é o servidor HTTP, que é um software capaz de entender URLs (endereços de sites na internet) e o protocolo HTTP. O protocolo HTTP (HyperText Transfer Protocol) é um conjunto de regras e padrões que define como os dados são transmitidos entre um cliente (geralmente um navegador web) e um servidor web na internet (MDN, 2025).

Finalmente, um navegador web (ou browser) é um aplicativo que permite aos usuários acessarem e interagirem com páginas e serviços web. (MDN, 2025) Os navegadores são instalados nos próprios dispositivos dos usuários. Os mais comuns são Google Chrome, Mozilla Firefox, Microsoft Edge e Safari.

Resumidamente, quando abrimos nosso navegador para buscar por determinado endereço da web, como https://www.wikipedia.org/, o navegador faz uma requisição dessa página ao servidor web usando o protocolo HTTP. Quando a requisição alcança o servidor web, este envia o arquivo solicitado requerido, também via HTTP. Pode soar estranho pensar que o servidor web nos fornece arquivo, mas nos tópicos seguintes, vamos entender como são constituídas as páginas da internet.

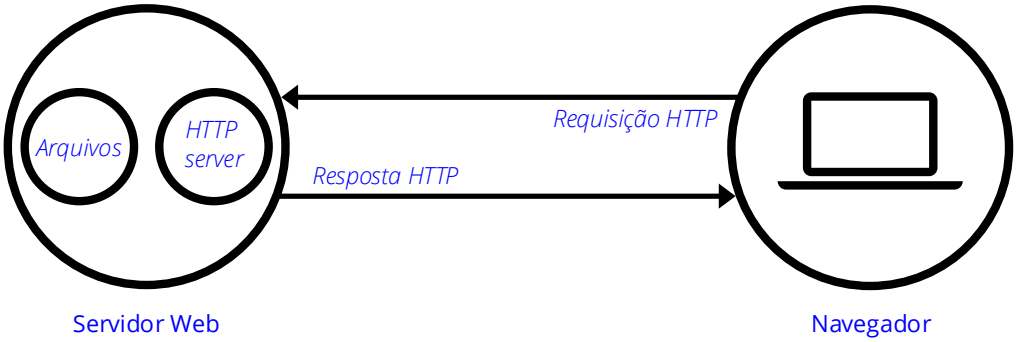


Figura 2. Representação esquemática do processo pelo qual um navegador solicita um arquivo a um servidor da web. Figura do site MDN Web Docs – Mozilla (MDN, 2025).

Uma página web é um documento que pode ser aberto em qualquer navegador que você tenha no seu computador. Esse documento é escrito, principalmente, em uma linguagem de marcação denominada HTML (que veremos na próxima seção). Todas as páginas web são alcançáveis através de um único endereço. Para acessar uma página, basta digitar seu endereço na barra de endereços do seu navegador. Um site é um conjunto de páginas web vinculadas (mais os recursos associados a elas) que compartilham um

único nome de domínio (endereço que permite acessar um site). Geralmente, um site possui muitas páginas. Por exemplo, o site da Wikipédia possui muitas páginas com diferentes temas.

Para entender a estrutura de páginas da internet, vamos supor que cada página que visitamos funciona como uma casa digital construída com diferentes materiais e camadas.

Figura 3. Imaginar a estrutura das páginas da internet como a de uma casa é uma analogia simples para entender como cada página é construída.

O HTML (Linguagem de Marcação de Hipertexto) é a estrutura básica das páginas da internet. "Hipertexto" refere-se aos links que conectam páginas da Web entre si, seja dentro de um único site ou entre sites. De maneira mais clara, os links são seções de uma página, que ao clicarmos, podem levar-nos a outras páginas ou seções. Ao carregar conteúdo na Internet e vinculá-lo a páginas criadas por outras pessoas, você se torna um participante ativo na world wide web (MDN, 2025). Assim como as paredes determinam as divisões dos cômodos em uma casa, o HTML define onde cada coisa está em uma página da internet: onde estão os textos, imagens, botões e links. Em termo formais, o HTML é é uma linguagem de marcação utilizada para dizer ao seu navegador como estruturar a página web que você visita.

O CSS é a decoração da casa. Assim, em um site, CSS é responsável pelo aspecto visual da página, incluindo cores, fontes, tamanhos e o posicionamento de cada elemento.

O JavaScript é a interatividade dos móveis da casa. Ele que permite que a página interaja com os usuários, faça cálculos, exiba animações ou carregue novos conteúdos.

Assim, quando abrimos qualquer página de um site, nosso navegador interpreta esses três elementos e nos mostra a página com sua aparência e funcionalidade. Para desempenhar atividades simples de raspagem de dados é necessário aprender um pouco sobre como navegar pela estrutura de HTML das páginas da internet. Atividades mais complexas de raspagem de dados também requerem conhecimento de JavaScript, mas não falarei dessas atividades avançadas neste e-book.

HTML

HTML não é uma linguagem de programação, visto que não trabalha com instruções de lógica, tal como Python, R ou JavaScript. O HTML permite que um desenvolvedor (um programador especializado em criar aplicativos e páginas web) crie e adicione estruturas (texto, imagem, vídeos) em uma página da internet.

Podemos dizer que o componente mais básico do HTML são os **elementos**. Por exemplo, a estrutura ** Seleção natural é massa ** representa um parágrafo de um texto. No geral, cada elemento possui uma **tag** de abertura e uma tag de fechamento. A tag de abertura consiste no nome do elemento (neste caso, *p* de parágrafo) envolvido por parênteses angulares, que é como chamamos este par de símbolos < >. A tag de fechamento delimita o final do elemento, e possui além dos parênteses angulares, uma barra antes do nome do elemento, </>. Neste exemplo, o conteúdo do elemento é a frase Seleção natural é massa. Portanto, as tags funcionam como containers, que nos informam algo sobre a informação que está em seu interior. No exemplo supracitado, a tag **p** informa que em seu interior há um parágrafo. (Duckett, 2011).

Os elementos podem conter **atributos**, que são informações extras sobre os elementos que nós não queremos que apareça no conteúdo real. Os atributos aparecem na tag de abertura de um elemento e são compostos de duas partes: um nome e uma valor, separados por um sinal de igual (Duckett, 2011). Por exemplo, na estrutura ** I am so happy. **, temos o atributo language com o valor en-us para indicar que este parágrafo está escrito em inglês dos Estados Unidos. Dois atributos bastante comuns são *id* e *class*. O *id* é um identificador de um elemento, e não pode ter um valor repetido dentro de uma página. O atributo class também é um identificador de

um elemento, mas pode ser repetido ao longo de um documento de HTML (Miranda, 2025).

Os elementos podem estar aninhados, ou seja, um elemento pode conter outro elemento em seu interior. Por exemplo, na estrutura ** Meu gatinho é muito mal-humorado. ,** usamos o elemento strong para deixar a palavra *muito* em negrito. Você precisa, no entanto, certificar-se de que seus elementos estejam adequadamente aninhados. No exemplo acima, abrimos primeiro o elemento , depois o elemento ; portanto, temos que fechar primeiro o elemento , depois o elemento (MDN, 2025). Devemos seguir esta lógica porque o elemento está contido no interior do elemento .

Alguns elementos não possuem conteúdo e, por isso, são chamados de elementos vazios. Observe esta estrutura: ****. O elemento <img não possui uma tag de fechamento do tipo </img> e nem possui um conteúdo interno. Em outras palavras, o conteúdo interno de um elemento img é a própria imagem. Observe a seguir um elemento img obtido de uma página da Wikipédia sobre regressão logística. Note que há uma série de atributos, tais como src (*source*, em inglês, que é o caminho que designa a origem da imagem) e outros atributos que determinam o comprimento e a largura da figura.

<img src="//upload.wikimedia.org/wikipedia/commons/thumb/a/ac/Logistic-curve.png/250px-Logistic-curve.png" decoding="async" width="250" height="188" class="mw-file-element" srcset="//upload.wikimedia.org/wikipedia/commons/a/ac/Logistic-curve.png 1.5x" data-file-width="320" data-file-height="240">

Para entender melhor como funcionam os blocos construtores das páginas da internet você pode visualizar alguns exemplos da estrutura de HTML em alguns sites. Nós podemos ver os códigos HTML que formam uma página de um site ao clicar com o botão direito do mouse sobre qualquer ponto da página e selecionar a opção Inspecionar. Observe, então, a caixa que surge ao lado direito superior da página. Essa caixa se chama Ferramentas de Desenvolvedor.

Os principais elementos de HTML que encontraremos na maioria das páginas web são:

<h1> até <h6>: Identifica títulos e subtítulos.

<body>: Abriga todo o conteúdo visível de uma página web (textos, imagens, vídeos) que vemos ao acessá-las usando um navegador.

<tittle>: Contém o nome da página web. Não confunda com a tag h1, por exemplo, que corresponde ao principal título ou cabeçalho de um texto.

<p>: Define cada um dos parágrafos de um texto.

: Listas não ordenadas, ou seja, listas nas quais a ordem dos elementos não tem importância, por exemplo, uma lista de compras.

: Listas ordenadas, por exemplo, um ordenamento de campeões de uma prova de esporte.

: Cada item de uma lista.

<a>: É o elemento usado para adicionar um link; este *a* é a forma abreviada de âncora. Geralmente, contém o atributo href (que significa referência em hipertexto), cujo valor é o endereço do link (outra página para onde o usuário será encaminhada).

Observe o exemplo abaixo retirado de um jornal chinês chamado China News Service. Eu fui até a seção de busca de notícias do site e digitei o termo *climate change* (mudanças climáticas em inglês). O resultado é uma página contendo com uma lista de títulos de notícias. Ao inspecionar a página, podemos encontrar o elemento que nos permite ter acesso ao texto de cada notícia. Podemos usar técnicas de raspagem de dados para buscar pelo elemento *a* e seu atributo *href* a fim de que possamos obter o texto completo de uma notícia.

Insights | UN peacekeeping chief calls for multilateralism to address global challenges, praising China's peacekeeping role

<img>: Exibe uma imagem.

<video>: Exibe um vídeo numa página.

<button>: Exibe um botão numa página. Pode ser usada em botões para enviar um formulário ou botões que permitem seguir para a página seguinte.

: Indica quebra de linha de um texto.

<b>: Deixa o texto em negrito.

: Deixa o texto em negrito e permite que um leitor com deficiência visual escute esta palavra ou trecho grifado com mais ênfase.

<i>: Formatação de texto em itálico.

: Formatação de texto em itálico interpretável por leitores de telas para pessoas com deficiência visual.

<u>: Permite sublinhar um texto.

<pre>: Permite a inserção de texto pré-formatado.

<code>: Permite a inserção de scripts de linguagem de programação.

: É um container genérico para divisão de conteúdo, que de certa forma não representa nada. Esse container pode ser estilizado usando CSS e JavaScript. Veja o exemplo abaixo para compreender melhor. No exemplo abaixo, a tag div agrupa uma imagem e um parágrafo. Podemos usar o atributo class e seu valor warning para adicionar estilos (cores, tamanho da fonte, borda) usando CSS.

Script de HTML:

```
<div class="warning">
  <img
    src="/shared-assets/images/examples/leopard.jpg"
    alt="An intimidating leopard." />
  <p>Beware of the leopard</p>
</div>
```

Script de CSS:

```
.warning {
  border: 10px ridge red;
  background-color: yellow;
  padding: 0.5rem;
```

```
  display: flex;
  flex-direction: column;
}

.warning img {
  width: 100%;
}

.warning p {
  font: small-caps bold 1.2rem sans-serif;
  text-align: center;
}
```

: É um container genérico em linha para conteúdo fraseado, que não representa nada por natureza. Ele pode ser usado para agrupar elementos para fins de estilo (usando os atributos class ou id), ou para compartilhar valores de atributos como lang. Veja o exemplo abaixo em que a tag span é utilizada para mudar o tamanho e a cor das fontes usadas em uma palavra:

```
<p> <span class="red">Python</span> é uma linguagem importante para analisar dados sobre esta espécie. </p>
```

Ser paciente e cauteloso ao inspecionar uma página será uma ferramenta fundamental para construir nossos primeiros rastreadores de páginas web. Tenha curiosidade de visitar diferentes sites e observar como os elementos de HTML podem ser diferentes para uma mesma estrutura, como títulos de notícias ou imagens.

CSS

O CSS (Cascading Style Sheets ou Folhas de Estilo em Cascata) é uma linguagem de estilo usada para descrever a apresentação de um documento escrito em HTML ou XML (outro tipo de linguagem de marcação que não estudaremos aqui) (MDN, 2025).A principal função dos códigos CSS é definir a aparência dos elementos de HTML que

estruturam uma página. Para visualizar elementos CSS de uma página devemos novamente clicar com o botão direito do mouse sobre qualquer ponto da página e selecionar a opção Inspecionar. Você pode observar que abaixo da caixa que contém os elementos HTML há outra caixa chamada Estilos. É aí que estão os elementos CSS de uma página.

O CCS é, principalmente, estudado por programadores frontend, que são aqueles responsáveis pelo aspecto visual das páginas web. Uma vez que a maioria dos projetos de web scraping busca coletar conteúdos ou informações (textos, datas, valores) de páginas web sem um interesse específico pelo visual da página, não é necessário ser um conhecedor de CSS para empreender projetos de web scraping.

JavaScript

JavaScript é a principal linguagem de programação usada para criar páginas web, visto que é fundamental para criar interatividade e funcionalidades dinâmicas em páginas da internet (MDN, 2025). Sempre que uma página web faz mais do que apresentar um conteúdo estático (textos e imagens), certamente, há uma estrutura de JavaScript subjacente. Isso acontece, por exemplo, quando adicionamos um produto a uma cesta de compras, escrevemos um comentário em uma rede social, clicamos em botão que permite ir para a página seguinte etc.

Geralmente, podemos encontrar elementos de JavaScript dentro da estrutura de HTML das páginas por meio da tag **script** na mesma janela onde encontramos os elementos de HTML ao inspecionar uma página. A compreensão dos códigos de JavaScript também não será necessária para os exercícios que iremos fazer neste e-book. Contudo, tenha em mente que coletar informações de sites complexos com muitas funcionalidades demanda ferramentas que possam interagir com elementos de JavaScript.

Referências

Duckett, Jon. HTML & CSS: Design and Build Websites. Indianapolis: John Wiley & Sons, 2011. ISBN 978-1-118-00818-8.

ECNS.CN search. Search ECNS, [s.d.]. Disponível em: https://search.ecns.cn/search.do. Acesso em: 25 set. 2025

MDN Contributors. HTML: Linguagem de Marcação de Hipertexto. MDN Web Docs, 27 abr. 2025. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>. Acesso em: 10 set. 2025.

MDN Contributors. *HTML básico — Aprendendo desenvolvimento web*. MDN Web Docs, 27 abr. 2025. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn_web_development/Getting_started/Your_first_website/Creating_the_content. Acesso em: 25 set. 2025.

Mozilla Contributors. Como a Internet funciona? — Aprendendo Desenvolvimento Web. MDN Web Docs. Última modificação em 27 de abril de 2025. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn_web_development/Howto/Web_mechanics/How_does_the_Internet_work. Acesso em: 13 out. 2025.

Mozilla Contributors. *O que é um servidor web (web server)?* — Aprendendo Desenvolvimento Web. MDN Web Docs. Última modificação em 27 de abril de 2025. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn_web_development/Howto/Web_mechanics/What_is_a_web_server. Acesso em: 13 de outubro de 2025.

Mozilla Contributors. *CSS (Folhas de Estilo em Cascata)* — MDN Web Docs. Última modificação em 24 de junho de 2025. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/CSS. Acesso em: 13 de outubro de 2025.

Mozilla Contributors. *O que é JavaScript?* — Aprendendo Desenvolvimento Web. MDN Web Docs. Última modificação em 23 de setembro de 2025. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn_web_development/Core/Scripting/What_is_JavaScript. Acesso em: 13 de outubro de 2025.

Miranda, Luiz Otávio; Malaquias, Tales Calogi. *Curso de JavaScript e TypeScript do básico ao avançado (JS/TS)*. Udemy. Disponível em: https://www.udemy.com/course/curso-de-javascript-moderno-do-basico-ao-avancado/. Acesso em: 25 set. 2025.

Wikipédia. Função sigmoide. Wikipédia, a enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_sigmoide. Acesso em: 25 ago. 2025.

Capítulo 2: Conhecendo o Beautiful Soup

Beautiful Soup é a biblioteca de Python que nos guiará nessa jornada de navegação pela internet. Ela é capaz de organizar a desordem das "sopas de letras" de elementos de HTML contidas em cada página. Para instalar o Beautiful Soup, será necessário abrir o terminal. O terminal é como uma janela de comunicação direta com o computador. Nesta janela, em vez de usar o mouse para abrir programas ou arquivos, nós precisamos digitar os comandos que queremos realizar. No nosso caso, o terminal será útil para instalar bibliotecas de Python essenciais para raspagem de dados.

Em Windows, para abrir o terminal, digite as teclas Windows + R. Escreva cmd e pressione Enter. No Mac, pressione Command + Espaço para abrir o Spotlight, digite Terminal e pressione Enter. No Linux, geralmente, se utiliza Ctrl + Alt + T ou Alt + F2 e pressiona Enter. Ao abrir o terminal (que, usualmente, se apresenta como uma tela preta) faça a instalação do Beautiful Soup usando o comando pip:

$ pip install beautifulsoup4

Ao instalar um pacote, você não precisa digitar o cifrão **$,** como está escrito acima. Esse símbolo é uma convenção que indica que essa linha de comando deve ser digitada no terminal.

Em seguida, pode ser necessário instalar um parser (também chamado de interpretador, em português). O parser é responsável por organizar códigos de HTML para que possamos extrair informações desejadas de maneira adequada. O Python já possui o parser HTML, mas há muitos tutoriais na internet que usam outros parsers, como o lxml ou o html5lib. Eu usarei neste e-book o parser HTML, mas é possível que, se você quiser seguir outros tutoriais, você precise instalar outros parsers da seguinte forma:

$ pip install lxml

$ pip install html5lib

Uma vez instalado o Beautiful Soup, é hora de dar os nossos primeiros passos. Nós também usaremos a biblioteca urllib, que faz parte da biblioteca padrão do Python, ou seja, não precisamos instalá-la. Essa biblioteca serve para abrir páginas da internet e obter os dados que queremos.

Vamos começar nosso primeiro código de raspagem de dados visitando uma notícia do site do Ministério do Meio Ambiente e Mudança do Clima do Brasil. Nossa primeira tarefa é simplesmente imprimir o título da notícia. Neste e-book, eu vou usar a fonte em azul para representar códigos que você deve escrever no Python ou os elementos de HTML que obtive de páginas web. Além disso, usarei fonte em cor vermelha para representar os resultados que o Python imprime. Ao inspecionar a estrutura de HTML do título de uma das notícias, vemos o seguinte:

<h1 property="rnews:headline" class="documentFirstHeading">Crianças e jovens debatem ações para impulsionar o desenvolvimento sustentável do país</h1>

Podemos ver que o título da notícia está contido dentro de uma tag h1, que é a tag que designa o título principal de uma página. Vamos fazer nosso primeiro código para tentar obter esse nome. Execute o seguinte código no Python:

```
from urllib.request import urlopen

from bs4 import BeautifulSoup

html = urlopen("https://www.gov.br/mma/pt-br/noticias/criancas-e-jovens-debatem-acoes-para-impulsionar-o-desenvolvimento-sustentavel-do-pais")

sopa = BeautifulSoup(html.read(), "html.parser")

print(sopa.h1)
```

Nas primeiras duas linhas, nós estamos apenas importando as bibliotecas necessárias para fazer nossa primeira raspagem de dados.

```
html = urlopen("https://www.gov.br/mma/pt-br/noticias/criancas-e-jovens-debatem-acoes-para-impulsionar-o-desenvolvimento-sustentavel-do-pais")
```

Dentro da função **urlopen**, eu colei o endereço da página da notícia que estamos estudando. Em Python, uma **função** é um bloco de código que executa uma tarefa específica. Neste caso, a tarefa da função urlopen é abrir o endereço da página que queremos analisar.

```
sopa = BeautifulSoup(html.read(), "html.parser")
```

Depois, usamos a função **BeautifulSoup**, com os métodos **html.read** e **html.parser** com o objetivo de organizar e interpretar a bagunça da “sopa de letras” dos

elementos de HTML da página. Então, criamos um objeto chamado sopa, onde nós salvamos o resultado dessa função. Agora, o objeto sopa contém todo a estrutura de HTML da página que fornecemos, mas de forma organizada. Assim, nós podemos buscar e manipular partes do HTML da página.

print(sopa.h1)

Finalmente, imprimimos na tela todo o elemento da tag h1. O resultado é o seguinte:

<h1 class="documentFirstHeading" property="rnews:headline">Crianças e jovens debatem ações para impulsionar o desenvolvimento sustentável do país</h1>

Não alcançamos o objetivo de imprimir o título da página, visto que queríamos apenas o seu conteúdo, o título da página. Para isso, vamos usar dois métodos: find e text.

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
html = urlopen("https://www.gov.br/mma/pt-br/noticias/criancas-e-jovens-debatem-acoes-para-impulsionar-o-desenvolvimento-sustentavel-do-pais")
sopa = BeautifulSoup(html.read(), "html.parser")
titulo = sopa.find("h1")
print(titulo.text)
```

Agora, depois de termos criado nossa sopa, criamos um objeto chamada titulo (evitamos usar acentos quando estamos escrevendo códigos em Python). Nesse caso, o objeto titulo guardará o resultado de tudo o que escrevemos à sua direita. O método find() encontra a tag que buscamos e seu conteúdo. Mas nós queremos apenas o seu conteúdo. Finalmente, precisamos fazer um pequeno ajuste no código, usando o atributo **.text** ou **.string**. Você também poderia usar o método **get_text()**. O atributo **.text** extrai apenas o conteúdo textual, eliminando as tags HTML.
Vale lembrar que, em Python, **um atributo** representa uma característica de um objeto (por exemplo, o texto contido em uma tag), enquanto **um método** é uma ação executada por esse objeto (como o find(), que busca uma tag específica).

Outra tarefa muito comum em raspagem de dados é quando queremos obter muitos elementos do mesmo tipo em uma página. Para exemplificar, visite a página Flora do Brasil e pesquise pelo gênero *Adiantum*, o gênero de plantas popularmente conhecidas

como avencas. Role com o mouse até onde há uma lista de espécies. Nosso desafio é obter os nomes de todas as espécies presentes na página. Ao inspecionar qualquer um dos nomes, vemos o seguinte elemento HTML:

```
<div class="taxon"> <i>Adiantum gracile</i></div>
```

Vemos aqui duas tags: **div** e **i**. A tag div é um elemento genérico que serve para agrupar outros elementos HTML em uma página. Neste exemplo, div contém o atributo **class** com o valor **taxon**. A tag i serve para aplicar estilo itálico ao texto. A seguir apresento um script para tentar obter os nomes de todas as espécies. Como o endereço da página é muito grande, eu optei por substitui-lo por “url da página”:

```
from urllib.request import urlopen

from bs4 import BeautifulSoup

html = urlopen (“url da página”)

sopa = BeautifulSoup(html.read(), "html.parser")

especies = sopa.find("div", {"class": "taxon"})

print(especies.text)
```

Note que neste caso não basta buscar apenas pela tag div, principalmente, porque esta tag é genérica e indica apenas um agrupamento de uma estrutura, que na página estudada é uma grande tabela com os nomes das espécies. Nós também precisamos especificar o atributo **class** e o valor **taxon**. Ao executar esse script conseguimos imprimir apenas o primeiro nome que aparece, que corresponde à família botânica Pteridaceae. Isso acontece porque o método **.find()** retorna apenas um elemento (geralmente, o primeiro) com as características buscadas. Vamos precisar usar o método **.find_all()** para obter todas as espécies da página. Veja uma reformulação do script a seguir:

```
from urllib.request import urlopen

from bs4 import BeautifulSoup

html = urlopen("link da página")

sopa = BeautifulSoup(html.read(), "html.parser")

especies = sopa.find_all("div", {"class": "taxon"})
```

```
for especie in especies:
    print(especie.text)
```

Observe que nessa resolução nós precisamos implementar um **loop for** para obter os nomes de todas as espécies. Um loop for é como uma instrução que pede ao Python para repetir uma ação para cada item em uma lista ou em uma sequência de valores. Neste caso, a ação que quisemos repetir várias vezes foi imprimir os nomes de todas as espécies presentes na página. Você deve ter observado que este método também imprime alguns nomes que não denominam espécies, como o nome da família botânica, visto que estes possuem a mesma estrutura de HTML. A exclusão dos nomes que não são espécies seria um pouco mais complicada. O importante é que aprendemos mais um método fundamental de Beautiful Soup.

Observe como podemos usar um código bastante semelhante para acessar uma página da Wikipédia e obter todos os links externos. Visite a página da Wikipédia do Brasil sobre o tema Aquecimento Global, role até o final, clique com o botão direito do mouse sobre qualquer trecho em azul nas referências bibliográficas e escolha a opção Inspecionar. Nós vemos o seguinte elemento de HTML:

```
<a rel="nofollow" class="external text"
href="https://www3.epa.gov/climatechange/basics/">"Climate Change: Basic
Information"</a>
```

Como já aprendemos, a tag **a** indica a presença de um link, ou seja, uma página externa. Neste exemplo, a tag **a** possui três atributos. O primeiro atributo é **rel**, cujo valor é **nofollow**, que possui o objetivo de evitar aumentar a relevância do link externo em buscadores da internet, como o Google. O segundo atributo é **class**, cujo valor é **external text**, cujo objetivo é indicar links externos na Wikipédia. Por fim, temos o atributo **href**, cujo valor é um link, ou seja, o endereço externo de outra página. Como queremos encontrar todos os links externos e não apenas o primeiro link que consta como valor do atributo href acima, o raciocínio será obter todas as tag **a** que contém o atributo **class** e o valor **external text**. Em seguida, vamos criar um loop for e usar o método **.get()** para obter os valores de todos os atributo **href**. Não usamos os métodos **.text()** ou **.get_text()** porque não queremos acessar um texto. Nós queremos acessar um valor, que é um link externo, que podemos clicar para acessar as respectivas páginas.

```
from urllib.request import urlopen

from bs4 import BeautifulSoup

html = urlopen("https://pt.wikipedia.org/wiki/Aquecimento_global")

sopa = BeautifulSoup(html.read(), "html.parser")

links_externos = sopa.find_all("a", {"class": "external text"})

for link in links_externos:

    href = link.get("href")

    print(href)
```

Apesar de ser um exemplo bastante interessante, você pode sentir-se frustrado se, ao tentar executá-lo obter como resposta uma mensagem de erro, tal como HTTP Error 403: Forbidden. O erro HTTP Error 403: Forbidden significa que o site (neste caso, a Wikipédia) bloqueou o acesso direto feito pelo urllib, porque ele não envia um cabeçalho de User-Agent. O User-Agent é uma identificação enviada por todo navegador ou cliente HTTP (como Chrome, Edge, Firefox, Safari etc.) quando ele faz uma requisição a um servidor da web. Em outras palavras, o servidor da Wikipédia detecta que a requisição veio de um script automatizado e não de um navegador. Vamos começar a aprender sobre este tema no capítulo 4.

Referências

Crummy, L. (s.d.). *Documentação Beautiful Soup* (versão pt-BR). Recuperado de https://www.crummy.com/software/BeautifulSoup/bs4/doc.ptbr/ — Acesso em: 14 de outubro de 2025.

Flora e Funga do Brasil. *Consulta: Adiantum* [site]. Disponível em: https://floradobrasil.jbrj.gov.br/consulta/?grupo=6&familia=null&genero=Adiantum&especie=&autor=&nomeVernaculo=&nomeCompleto=&formaVida=null&substrato=null&ocorreBrasil=QUALQUER&ocorrencia=OCORRE&endemismo=TODOS&origem=TODOS&regiao=QUALQUER&ilhaOceanica=32767&estado=QUALQUER&domFitogeograficos=QUALQUER&vegetacao=TODOS&mostrarAte=SUBESP_VAR&opcoesBusca=TODOS_OS_NOMES&loginUsuario=Visitante&senhaUsuario=&contexto=consulta-publica&pagina=1#CondicaoTaxonCP. Acesso em: 14 de outubro de 2025.

Ministério do Meio Ambiente e Mudança do Clima. Crianças e jovens debatem ações para impulsionar o desenvolvimento sustentável do país. Publicado em: 08 de outubro de 2025, 14h31. Disponível em: https://www.gov.br/mma/pt-br/noticias/criancas-e-jovens-debatem-acoes-para-impulsionar-o-desenvolvimento-sustentavel-do-pais. Acesso em: 14 de outubro de 2025.

Wikipédia contributors. *Aquecimento global*. Wikipédia, a enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/Aquecimento_global. Acesso em: 14 de outubro de 2025.

Capítulo 3: Navegando por documentos de HTML usando hierarquias

No capítulo anterior nós aprendemos como encontrar e obter determinados conteúdos de uma página de internet por meio da análise de seus elementos de HTML. Nós buscamos os elementos que desejávamos por meio de tags e seus respectivos atributos e valores. Contudo, também é possível navegar pela estrutura de códigos de HTML baseando-se nas relações hierárquicas das tags.

Documentos HTML frequentemente possuem tags inseridas dentro de outras tags. Dessa forma, é comum vermos tutoriais na internet falando sobre árvores de HTML, e sobre como navegar por seus ramos. Para aprender como navegar em árvores de HTML, vou trazer aqui um resumo da documentação da biblioteca Beautiful Soup disponível na internet em português (Crummy, 2025). Essa documentação usa como exemplo didático um documento de HTML que é um trecho do livro de Alice no País das Maravilhas em inglês. Observe este documento a seguir:

```
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""
```

Copie esse documento e cole em um arquivo de Python. Em seguida, execute o script abaixo para visualizar melhor a estrutura HTML deste documento. Após criar a sopa, nós usaremos o método prettify(), cujo objetivo é modificar o documento de HTML e apresentá-lo de maneira organizada e legível, incluindo a presença de indentação (espaçamento na margem esquerda) e quebras de linha.

```
from bs4 import BeautifulSoup
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""
sopa = BeautifulSoup(html_doc, 'html.parser')
print(sopa.prettify())
```

Ao executarmos este código podemos ver o documento HTML após ser organizado para melhor visualização:

```
<html>
 <head>
  <title>
   The Dormouse's story
  </title>
 </head>
 <body>
  <p class="title">
   <b>
    The Dormouse's story
```

```
  </b>
 </p>
 <p class="story">
  Once upon a time there were three little sisters; and their names were
  <a class="sister" href="http://example.com/elsie" id="link1">
   Elsie
  </a>
  ,
  <a class="sister" href="http://example.com/lacie" id="link2">
   Lacie
  </a>
  and
  <a class="sister" href="http://example.com/tillie" id="link3">
   Tillie
  </a>
  ;
and they lived at the bottom of a well.
 </p>
 <p class="story">
  ...
 </p>
</body>
</html>
```

Nós já aprendemos a maneira mais simples de navegar por uma árvore de HTML, que é buscando exatamente as tags que desejamos. Por exemplo, vamos imprimir o elemento título deste documento:

```
print(sopa.title)
<title>The Dormouse's story</title>
```

Também já aprendemos que é possível descer ou escavar uma árvore de HTML especificando a relação de hierarquia entre duas tags. Por exemplo, vamos obter a primeira tag **b** que aparece depois da tag **body**. A tag **body** contém todo o conteúdo visível (para um usuário) de uma página web, já a tag **b** estabelece que seu conteúdo estará em negrito:

```
print(sopa.body.b)
<b>The Dormouse's story</b>
```

Por fim, nós também já vimos casos em que é possível buscar todas as tags de um tipo em um determinado documento de HTML usando o método **.find_all()**. Vamos encontrar, por exemplo, todas as tags **a** do documento estudado. O resultado será um objeto do tipo **bs4.element.ResultSet** e que se comporta como uma lista (uma coleção de objetos):

```
print(sopa.find_all('a'))

[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Após revisarmos como encontrar partes de um documento de HTML guiando-se pelos nomes das tags e seus atributos e valores, vamos começar a investigar como navegar por uma árvore de HTML usando especificamente uma lógica de hierarquia. Nós podemos buscar as filhas de uma determinada tag usando **.contents**. Uma tag filha é uma tag que vem logo abaixo de outra em uma estrutura hierárquica de um documento de HTML. Por exemplo, no documento que estamos estudando a tag **head** é uma filha da tag **html**, da mesma forma que **title** é uma filha da tag **head**. Vamos usar um exemplo

para compreender melhor esse processo. Vamos criar um objeto para armazenar a tag head e vamos imprimir esse objeto:

```
head_tag = sopa.head
print(head_tag)
<head><title>The Dormouse's story</title></head>
```

O resultado é um cabeçalho do documento HTML, o qual contém seu título dentro da tag title. Em seguida, vamos usar **.contents** para imprimir a tag **title**, que é a tag filha de **head**:

```
print(head_tag.contents)
[<title>The Dormouse's story</title>]
```

Um ponto que precisamos ter atenção é que em Beautiful Soup **.contents** retorna todos os filhos diretos de uma tag. Esses filhos incluem tags, strings (textos presentes dentro de tags) e até mesmo quebras de linha, que em Python são representadas por **\n**. Vamos usar outro exemplo com esse mesmo documento para entender melhor. Vamos criar uma lista contendo todas a tag **p** com o atributo **class** e o valor **story.** Em seguida, vamos usar acessar os elementos filhos dessa tag **p**. Execute o código abaixo:

```
sopa = BeautifulSoup(html_doc, "html.parser")
lista = sopa.find("p", {"class": "story"})
tags_filhas = lista.contents
print(tags_filhas)
['Once upon a time there were three little sisters; and their names were\n', <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>, ',\n', <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>, ' and\n', <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>, ';\nand they lived at the bottom of a well.']
```

O resultado é uma lista cujos elementos estão separados por vírgula. Uma lista em Python é um tipo de dado que permite armazenar múltiplos itens em uma única variável. Esses itens podem ser de qualquer tipo (números inteiros, números decimais ou strings) e podem ser organizados em uma ordem específica. Como no exemplo acima, esses itens são organizados dentro de colchetes. Observe que a estrutura que está abaixo da tag p contém string (texto), quebra de linha representada por **\n**, e várias tags com seus atributos e valores. Portanto, esse exemplo reforça que o uso de **.contents** permite acessar toda a estrutura que está abaixo de uma determinada tag.

Agora, em vez de imprimir todos esses filhos, nós podemos querer imprimir apenas os nomes das três personagens contidos na lista acima: Elsie, Lacie e Tillie. Esses nomes estão dentro da tag **a** com o atributo **class** e o valor **sister**.

```
tags_a = lista.find_all("a", {'class': 'sister'})
for tag in tags_a:
   print(tag.get_text())
```

```
Elsie
Lacie
Tillie
```

Nós podemos agora fazer o caminho inverso. Em um documento HTML, toda tag e toda string contém tags mães, que são tags que estão imediatamente acima. Essas tags são chamadas de **parents** (pais, em inglês). No documento estudado, já vimos que a tag **title** é filha da tag **head**. Logo, a tag **head** é mãe de **title**. Assim, podemos acessar **head** a partir de **title**:

```
title_tag = sopa.title
print(title_tag.parent)
```

```
<head><title>The Dormouse's story</title></head>
```

Além de navegar para baixo buscando por tags filhas e para cima buscando por tags mães, também é possível navegar para os lados buscando por tags irmãs. As tags irmãs (ou siblings, em inglês) são tags que estão num mesmo nível hierárquico. Para compreender como podemos navegar para o lado em uma árvore de HTML, vamos usar como exemplo o seguinte documento que também está disponível na página do Beautiful Soup. Execute o código a seguir para observar o documento HTML organizado:

```
sibling_soup= BeautifulSoup("<a><b>text1</b><c>text2</c></b></a>", 'html.parser')
print(sibling_soup.prettify())
```

```
<a>
 <b>
  text1
 </b>
 <c>
  text2
 </c>
</a>
```

No documento acima, as tags **b** e **c** estão no mesmo nível, ambas são filhas da tag **a**. Quando um documento de HTML é impresso usando o método **.prettify()**, as tags irmãs aparecem no mesmo nível de indentação, ou seja de espaçamento. Para duas tags serem consideradas irmãs não basta que elas estejam no mesmo nível hierárquico, elas também precisam ser descendentes da mesma tag mãe. Nós podemos usar **.next_sibling** para mover-nos para frente e obter os elementos irmãos que está no mesmo nível hierárquico:

```
print(sibling_soup.b.next_sibling)
<c>text2</c>
```

Observando o documento de HTML, o que nós fizemos foi ter como ponto de partida a tag **b**. Então, nós nos movemos para o elemento seguinte que está no mesmo nível. Este é a tag **c** e o seu conteúdo. Podemos agora fazer o caminho inverso. Vamos tomar como ponto de partida a tag **c** e mover-nos para trás buscando o elemento anterior que está no mesmo nível, que é a tag **b** com o seu conteúdo. Para fazer isso, vamos usar **.previous_sibling**:

```
print(sibling_soup.c.previous_sibling)
<b>text1</b>
```

A tag **b** não possui **.previous_sibling** porque não há nada antes dela no mesmo nível hierárquico. Assim, se tentarmos imprimir seu irmão anterior, o resultado é **None**, um indicativo de que o elemento que estamos buscando não existe.

```
print(sibling_soup.b.previous_sibling)
None
```

Em documentos reais, **.next_sibling** ou **.previous_sibling** de uma tag geralmente são strings (texto) contendo espaços em branco. Vamos voltar a analisar o documento do início do capítulo, que é um trecho do livro de Alice no país das maravilhas:

```
<html>
 <head>
  <title>
   The Dormouse's story
  </title>
```

```
</head>
<body>
 <p class="title">
  <b>
   The Dormouse's story
  </b>
 </p>
 <p class="story">
  Once upon a time there were three little sisters; and their names were
  <a class="sister" href="http://example.com/elsie" id="link1">
   Elsie
  </a>
  ,
  <a class="sister" href="http://example.com/lacie" id="link2">
   Lacie
  </a>
  and
  <a class="sister" href="http://example.com/tillie" id="link3">
   Tillie
  </a>
  ;
and they lived at the bottom of a well.
 </p>
 <p class="story">
```

```
  ...
 </p>
</body>
</html>
```

Vamos imprimir a .next_sibling da primeira tag a:

```
sopa = BeautifulSoup(html_doc, 'html.parser')
tag_a = sopa.a
print(tag_a.next_sibling)
```

```
,
```

O resultado impresso é apenas uma vírgula e uma quebra de linha. Um raciocínio comum seria pensar que o **next_sibling** da primeira tag **a** seria a seguinte tag **a**. Contudo, observe o documento acima e veja o seguinte elemento na mesma altura da tag **a** é uma vírgula, seguida de uma quebra de linha. Nós podemos também imprimir todos os elementos irmãos que vem depois da primeira tag a:

```
for sibling in tag_a.next_siblings:
   print(repr(sibling))
```

```
',\n'
<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
' and\n'
<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
';\nand they lived at the bottom of a well.'
```

Neste exemplo, nós usamos a função **repr()** dentro da função **print()**. A função **repr()** em Python retorna a representação "oficial" de um objeto, ou seja, uma versão textual que mostra como o Python "vê" esse objeto internamente. Isso inclui, por exemplo, a estrutura **\n**.

Este capítulo ilustra outra abordagem que podemos usar para encontrar conteúdos que desejamos dentro de uma página. Sempre que possível, tente ler a documentação do Beautiful Soup para tirar suas dúvidas e tente executar os scripts apresentados aqui.

Referências

Crummy, L. (s.d.). *Documentação Beautiful Soup* (versão pt-BR). Recuperado de https://www.crummy.com/software/BeautifulSoup/bs4/doc.ptbr/ — Acesso em: 14 de outubro de 2025.

Capítulo 4: Conectando adequadamente e lidando com exceções

No mundo real, diferentemente dos exemplos didáticos, as páginas da internet podem ser bagunçadas, mal formatadas ou até mesmo deixar de existir (Mitchell, 2024). Ao aprendermos técnicas de raspagem de dados precisamos saber lidar com alguns problemas desse tipo que podem afetar nosso trabalho ou estudo. Vamos executar códigos reais para entender esse problema. Execute o seguinte no Python e verifique a mensagem de erro que é impressa:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
html = urlopen("https://www.globo.com/esta_pagina_nao_existe")
raise HTTPError(req.full_url, code, msg, hdrs, fp)

urllib.error.HTTPError: HTTP Error 404: Not Found
```

O que aconteceu aqui é que nós tentamos acessar uma URL válida (um site que existe), mas cuja página específica não existe. Nesta situação, a função **urlopen()** vai retornar um **erro de HTTP**. Esse erro pode ser "404 Page Not Found", "500 Internal Server Error" ou ainda "403 – Forbidden". Os dois primeiros exemplos podem referir-se a erros de digitação ou à busca por páginas que não existem mais, enquanto o último erro refere-se a páginas cujo acesso não é permitido.

Vejamos agora o segundo tipo de erro que pode quebrar a execução de códigos de raspagem de dados:

```
html = urlopen("http://www.esseDominioNaoExiste.com")
raise URLError(err)

urllib.error.URLError: <urlopen error [Errno 11001] getaddrinfo failed
```

Neste caso, nós tentamos abrir uma URL que não existe, o que faz com que a função **urlopen()** retorne um **erro de URL**. Esse tipo de erro também pode ocorrer se o servidor está fora do ar ou inacessível (por exemplo, um servidor em manutenção), se a URL está mal formatada ou não segue o formato correto (por exemplo, se você esquecer de digitar http:// no início do endereço desejado) ou ainda se houver falhas na conexão à internet durante a execução do código.

Para contornar a possibilidade de existências de qualquer um desses dois erros, e evitar que um código de raspagem de dados falhe por causa deles, nós teremos que de adicionar algumas poucas linhas que lidam com esses erros logo no início do código:

```
from urllib.request import urlopen
from urllib.error import HTTPError
from urllib.error import URLError
from bs4 import BeautifulSoup

try:
    html = urlopen('http://www.pythonscrapingestaurlnaoexiste.com')
except HTTPError as e:
    print("Erro de HTTP")
except URLError as e:
    print("O servidor não pôde ser encontrado.")
else:
    print("Funcionou!")
O servidor não pôde ser encontrado.
```

Observe que agora as quatro primeiras linhas do nosso código estão diferentes. Como já vimos anteriormente, urllib é uma biblioteca padrão do Python, ou seja, nós precisamos apenas importá-la, sem necessidade de instalar previamente. Nós já usamos em vários exemplos o módulo urllib.request, que é usado para abrir e fazer requisições para URLs. Agora, estamos também usando o módulo urllib.error, que serve para lidar com possíveis erros ao tentar acessar URLs. O módulo urllib.error possui duas principais classes, HTTPError e URLError, as quais lidam com erros de HTTP e de URL respectivamente.

Em seguida, nós usamos uma estrutura lógica, um bloco **Try-Except**. O uso da palavra try (verbo tentar, em inglês) solicita ao Python que abra a página contida dentro da variável html. Se o site existe, mas há algum problema na página que possa gerar um erro de HTTP, o Python não acessa a página e imprime **Erro de HTTP**. Descendo mais no código, se o endereço fornecido for inválido ou o servidor não puder ser encontrado (por exemplo, porque o site não existe), o Python imprime **O servidor não pôde ser encontrado**. Portanto, essa estrutura lógica **Try-Except** permite ao mesmo tempo tentar

acessar uma URL, enquanto verifica a possibilidade de erros de HTTP e de URL. Veja um exemplo com uma página real abaixo:

```
from urllib.request import urlopen
from urllib.error import HTTPError
from urllib.error import URLError
from bs4 import BeautifulSoup
try:
    html = urlopen('https://revistagalileu.globo.com/um-so-planeta/noticia/2024/12/cientistas-colocaram-cameras-em-ursos-de-oculos-por-4-meses-imagens-impressionam.ghtml')
except HTTPError as e:
    print("Erro de HTTP")
except URLError as e:
    print("O servidor não pôde ser encontrado.")
else:
    print("Funcionou!")

sopa = BeautifulSoup(html.read(), "html.parser")
titulo = sopa.find("h1", {"class":"content-head__title"})

if titulo:
    print(titulo.text)
else:
    print("Título não encontrado.")
```

Funcionou!

Cientistas colocaram câmeras em ursos-de-óculos por 4 meses; imagens impressionam

Neste caso, eu visitei a página de uma notícia científica no site do Globo com o objetivo de imprimir o seu título. Observe que além da verificação de possíveis erros de HTTP ou de URL, eu fiz uma verificação adicional. Eu usei a estrutura lógica **if-else** (if significa “se” em inglês) para que verifique se o elemento titulo que estamos buscando, de fato, existe. Se não existisse, o código iria imprimir a frase Título não encontrado.

No segundo capítulo, ao tentar obter links externos de uma página da Wikipédia, nós obtemos o erro “HTTP Error 403: Forbidden”. Este erro, geralmente, acontece porque muitas páginas web possuem estruturas que impedem acesso automatizado (acessos que não são feitos por seres humanos) (Omisola, 2025). Uma das soluções para este problema é adicionar o User-Agent no script. O User-Agent é uma identificação enviada por todo navegador ou cliente HTTP (como Chrome, Edge, Firefox, Safari etc.) quando ele faz uma requisição a um servidor da web. Essa identificação diz ao servidor quem está pedindo a página — ou seja, que tipo de software está acessando o site. Para descobrir seu User-Agent, você pode visitar a página: https://www.whatismybrowser.com/. Agora, vejamos como o script da Wikipédia ficaria corrigido:

```
import requests
from bs4 import BeautifulSoup

url = "https://pt.wikipedia.org/wiki/Aquecimento_global"
headers = {"User-Agent": "SEU USER AGENT"}

response = requests.get(url, headers=headers)
sopa = BeautifulSoup(response.text, "html.parser")

links_externos = sopa.find_all("a", attrs={"class": "external text"})

for link in links_externos:
   print(link.get("href"))
```

No entanto, esta solução pode ser provisória. Se você tenta fazer raspagem de dados com frequência em determinada página web, mesmo que identifique seu User-Agent e seu IP, você pode novamente ser bloqueado. Se seus projetos estão ficando maiores, você precisa começar a estudar diversas técnicas de superar ou driblar este bloqueio. Mas não se esqueça. Reflita sempre sobre a legalidade e o bom uso de seu projeto. Se você está sendo bloqueado por alguma página web, é possível que esteja fazendo excessivas visitas a essa página, o que pode sobrecarregar o servidor do site em questão.

Referências

Almeida, Arthur. Cientistas colocaram câmeras em ursos-de-óculos por 4 meses; imagens impressionam. Revista Galileu — Um Só Planeta. 06 de dezembro de 2024. Disponível em: https://revistagalileu.globo.com/um-so-planeta/noticia/2024/12/cientistas-colocaram-cameras-em-ursos-de-oculos-por-4-meses-imagens-impressionam.ghtml . Acesso em: 16 de outubro de 2025.

Mitchell, R. 2024. Web Scraping com Python: Coletando dados da web moderna. O'Reilly. 3 ed. Novated Editora LTDA.

Omisola, Idowu. *Error 403 in Web Scraping: 7 Easy Solutions* — ZenRows Blog. Atualizado em 26 de setembro de 2024. Disponível em: https://www.zenrows.com/blog/403-web-scraping#set-fake-user-agent. Acesso em: 16 de outubro de 2025.

Wikipédia contributors. *Aquecimento global*. Wikipédia, a enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/Aquecimento_global. Acesso em: 16 de outubro de 2025

Capítulo 5: Usando funções para automatizar a raspagem de dados

Este capítulo tem a intenção de mostrar como podemos usar funções para automatizar e, consequentemente, acelerar o processo de raspagem de dados. Funções são sequências nomeadas de instruções que executam uma operação de computação (Downey, 2018). Criar funções é uma das habilidades mais importantes em programação porque permite que executemos uma determinada tarefa muitas vezes em diferentes contextos sem necessidade de rescrever o código. Fazendo uma analogia com algo do cotidiano é como se fosse o uso de massa pronta para preparar uma receita de um bolo. Você economiza muito tempo, conhecimentos e habilidades ao usar um material que já está pronto para essa determinada tarefa.

A criação de uma função em Python começa ao digitarmos **def** seguido do nome que queremos dar à função e, em seguida, escrevemos o código que queremos automatizar. Uma função pode ter **parâmetros** e **argumentos** (Downey, 2018). Se pensarmos na analogia do bolo, os ingredientes presentes na lista da receita são os parâmetros, ou seja, aquilo que uma função precisa ter para funcionar. Já os argumentos são os ingredientes reais que você busca na sua dispensa para o preparo. Em função, os argumentos são os valores reais que são passados para uma função.

Vamos criar nossa primeira função abaixo, cujo nome será saudar_pessoas. Observe que ao denominar uma função, não devemos usar pontuações e nem espaços. Se quisermos usar mais de uma palavra, usamos o símbolo de sobrescrito. Em seguida, adicionamos um parâmetro entre parênteses (aqui usamos a palavra nome). Logo abaixo, com uma indentação e com aspas triplas adicionamos um pequeno texto que explica o que é esta função. Isso é importante porque outras pessoas podem precisar de nossos códigos no futuro. E, finalmente, abaixo, vem o corpo da função, que determina o que ela faz. Nossa primeira função simplesmente imprime uma saudação e o nome da pessoa.

```
def saudar_pessoas(nome):
    '''
    Essa função serve para saudar uma pessoa
    '''
    print("Olá " + nome)
```

Depois de criar nossa função, nós vamos executá-la. Para isso, nós vamos escrever o nome da função e, dentro dos parênteses, vamos substituir o parâmetro nome por outro nome qualquer, que será um argumento, como vimos anteriormente. É importante ressaltar que este nome precisa ser escrito entre aspas para que o Python entenda que se trata de um texto. Logo, veremos casos em que nomes não precisam ser escritos entre aspas, e ficará claro para você em que situações isso é necessário.

```
saudar_pessoas("Charles Darwin")
Olá Charles Darwin
```

Nem sempre uma função precisa de um parâmetro. Lembre-se de jogos de vídeo games que começam com um tipo de saudação. Vamos criar abaixo uma função que apenas pede para o jogador pressionar start para começar o jogo. Observe que essa função sempre executa a mesma frase.

```
def iniciar():
    '''
    Essa função instrui o jogador a iniciar.
    '''
    print("Pressione start para começar.")

iniciar()
Pressione start para começar.
```

As funções também são muito úteis para operações matemáticas.

```
def quadrado(numero):
    '''
    Calcula o quadrado de um número.
    '''
    print(numero * numero)

quadrado(5)
25
```

Até agora nós criamos funções que imprimem algo em tela, mas esse não é uso mais comum das funções. Normalmente, os programadores definem funções que retornam algum valor, o qual pode ser armazenado em uma variável, e posteriormente utilizado. Para criar uma função que retorna um valor precisamos usar a palavra-chave **return** no seu interior. Veja abaixo o exemplo de uma função chamada multiplicar. Essa função possui dois parâmetros (numero1 e numero2). Ao usarmos essa função, nós substituímos os parâmetros por quaisquer dois números. No exemplo abaixo, nós chamamos a função para multiplicar 2 por 10. Observe que nós salvamos o resultado dessa multiplicação numa variável chamada resultado. Agora, podemos usar essa variável resultado para qualquer coisa, incluindo outras operações matemáticas, como fizemos abaixo.

```
def multiplicar(numero1, numero2):
'''
   Calcula o produto de dois números
 '''
   total = numero1 * numero2
   return total

resultado = multiplicar(2, 10)

print(resultado /10)
2
```

Na função anterior, nós usamos os argumentos numero1 e numero2 para deixar claro para o usuário que são necessários dois argumentos para o funcionamento adequado. Contudo, por vezes, não temos em mente quantos argumentos podem ser necessários para criar uma função. Para deixar claro que não há um limite quanto ao número de argumentos que uma função pode receber podemos criar uma função usando em seu interior *args. Veja a função abaixo que faz um somatório quaisquer quantidade de números que um usuário inserir.

```
def somatorio(*args):

  return sum(args)
```

```
valor_total = somatorio(5, 10, 2, 20)

print(valor_total)
```

Agora, estamos prontos para criar uma função getTitle (obter título), que é capaz de obter títulos de quaisquer página da internet.

```
import ssl
from urllib.request import urlopen
from urllib.error import HTTPError
from bs4 import BeautifulSoup

# Criar contexto SSL que ignora verificação de certificado
ssl_context = ssl._create_unverified_context()

def getTitle(url):

    # Tenta abrir uma página e obter o arquivo HTML
    # Se der erro, retorna None

    try:
        html = urlopen(url, context=ssl_context)
    except HTTPError as e:
        return None
    except Exception as e:
        print("Erro ao acessar:", e)
        return None

    # Tenta obter uma tag da página
    # Se der erro, retorna None

    try:
```

```
        bs = BeautifulSoup(html.read(), 'html.parser')
        title = bs.body.h1
    except AttributeError:
        return None

    return title

title = getTitle(url da pagina')

if title is None:
    print("O título não pôde ser encontrado.")
else:
    print(title)
```

Não se assuste se o código de uma função parece longo. Você precisa cria-lo apenas uma vez. Agora, sempre que quiser obter o título de uma página, basta usar title = getTitle(‘url da página’). Nesta função há algo novo para você, o uso do contexto SSL. Nós criamos um contexto SSL que ignora a verificação do certificado do servidor, permitindo acessar páginas HTTPS mesmo quando o certificado é inválido ou autoassinado — não recomendado em produção (projetos reais grandes). Um certificado digital é um arquivo usado por sites com HTTPS para provar sua identidade e garantir uma conexão segura e criptografada entre o servidor e o navegador.

Agora que você sabe o básico sobre HTML, funções (incluindo as principais funções da biblioteca Beautiful Soup), e sabe tratar exceções, no próximo capítulo vamos mergulhar em um projeto real um pouco mais longo.

Referências

Downney, Allen B. 2018. Pense em Python : Pense como um cientista da computação. Tradução de Sheila Gomes. 2. ed. São Paulo: Novatec, 2018. ISBN 978-85-7522-508-0.

Capítulo 6: Escrevendo web crawlers

Usamos a nomenclatura web crawlers (rastreadores web) para nos referir a scripts de raspagem de dados que percorrem várias páginas web diferentes. Por exemplo, você pode implementar um script em Python para buscar notícias sobre mudanças climáticas em diferentes sites de notícias. Seu objetivo seria obter o texto completo de várias notícias de um site, uma vez obtido uma quantidade *x* de notícias, você passaria para outro site, até alcançar o objetivo desejado (Mitchell, 2024).

Antes de seguir, quero te apresentar outro aspecto do Python que é bastante útil em projetos de raspagem de dados: as **expressões regulares**. Uma RegEx ou expressão regular é uma sequência de caracteres que tem um padrão específico (W3Schools, 2025). Elas podem ser usadas para detectar a presença ou a ausência de uma palavra ou número ao comparar com um determinado padrão estabelecido pelo programador. Para estudar expressões regulares, precisamos importar a biblioteca **re**, que é nativa do Python, ou seja, não é necessário fazer instalação previamente. Não quero trazer aqui uma grande tabela mostrando todas as expressões regulares de Python. Isso é facilmente encontrado na internet. Trago apenas alguns exemplos práticos apresentados por Frederico Garay no curso de Python Total da Udemy, o qual recomendo aos interessados em aprender programação básica em Python.

Primeiramente, observe como usamos a função **re.search()** da biblioteca **re** para encontrar uma palavra em um texto.

```
import re
texto = "Se necessita de ajuda, chame ao serviço de ajuda online (081) 98625-9900, funcionamos 24 horas"
padrao = "ajuda"
busca = re.search(padrao, texto)
print(busca)
<re.Match object; span=(15, 20), match='ajuda'>
```

Neste exemplo, o resultado da função é uma confirmação de que a palavra ajuda existe no texto, e a apresentação das posições de início e fim da primeira aparição desta palavra. Se substituirmos **re.search()** por **re.findall()**, receberemos uma lista contendo a

palavra ajuda duas vezes, visto que esta palavra aparece duas vezes. Agora, para um primeiro exemplo concreto usando expressões regulares, nós vamos usar **\d** (expressão que denomina um número) dentro da expressão **r"** para verificar se existe um número de telefone no texto. Se existir o padrão desejado, ele será impresso.

```
import re
texto = "Chama ao número 564-524-6500 agora mesmo."
padrao = r'\d\d\d-\d\d\d-\d\d\d\d'
resultado = re.search(padrao, texto)
print(resultado)
<re.Match object; span=(16, 28), match='564-524-6500'>
```

Esse script pode ser simplificado para deixar mais claro a quantidade de números que pode ter em cada parte de um número telefônico. Poderíamos escrever:

```
padrao = r'\d{3}-\d{3}-\d{4}'
```

Vamos criar agora outro exemplo interessante que verifique se uma senha tem oito dígitos, sendo sete alfanuméricos (números ou letras), mas que o primeiro elemento não seja um número. Vamos usar a expressão regular **\D** para verificar que algo não é um número. Ou seja, para verificar o inverso de ser um número **\d** usamos a mesma letra em maiúsculo. A expressão regular que verifica qualquer caractere alfanumérico (letras ou números) é **\w**.

```
import re
senha = 'paulo645'
padrao = r'\D{1}\w{7}'
checar = re.search(padrao, senha)
print(checar)
<re.Match object; span=(0, 8), match='paulo645'>
```

Outro aspecto importante das expressões regulares é o uso de operadores especiais. Vamos começar com um exemplo do operador barra em pé | que significa a expressão **ou**. No exemplo a seguir buscaremos no texto as palavras sábado ou domingo.

```
import re
texto = "Não atendemos domingo"
```

```
buscar = re.search(r'sábado|domingo', texto)
print(buscar)
```

```
<re.Match object; span=(14, 21), match='domingo'>
```

Usamos o operador acento circunflexo ^ para verificar se um padrão se encontra no começo de um string. Podemos querer verificar, por exemplo, se a primeira letra de uma senha não é um número. Uma vez que estou usando um número na primeira posição, o resultado será None. Ou seja, o primeiro elemento do string é um número.

```
import re
senha = "5Abracadabra"
buscar = re.search(r'^\D', senha)
print(buscar)
```

```
None
```

O operador cifrão **$**, verifica se um padrão se encontra no final de um string. Usando o mesmo exemplo, podemos querer verificar se o último elemento da senha não é um número. Como, de fato, não é um número, obtemos tal último elemento.

```
import re
senha = "5Abracadabra"
buscar = re.search(r'\D$', senha)
print(buscar)
```

```
<re.Match object; span=(11, 12), match='a'>
```

Podemos também criar uma função que permita verificar se algo digitado é um e-mail válido. O operador + significa que algo deve aparecer uma ou mais vezes. No padrão abaixo queremos verificar se existe um @ seguido de, pelo menos um caractere alfanumérico. Veja abaixo como seria:

```
import re


def verificar_email(email):
    padrao = r'@\w+\.com'
    verificar = re.search(padrao, email)
    if verificar:
        print("O endereço de email está correto.")
```

```
    else:
        print("O endereço de email está incorreto.")
```

Uma vez que você tenha aprendido um pouco sobre expressões regulares em Python, podemos seguir com nossa atividade de rastreio da web. A atividade que apresento é proposta por Ryan Mitchell em seu capítulo 6 do livro Web Scraping com Python. O objetivo é percorrer páginas da Wikipédia para obter todos os links de artigos internos contidos em uma dada página. No capítulo 4, nós vimos como obter links externos presentes no final de uma página da Wikipédia. Agora, nosso objetivo será obter links internos, ou seja, outras páginas de artigos dentro da própria Wikipédia.

Veja abaixo um script para obter todos os links quaisquer presentes em uma página da Wikipédia sobre a atriz Fernanda Torres. Lembre-se que um link é caracterizado por ter uma tag **a** com o atributo **href**. Observe também que nós adicionamos um User Agent para evitar o erro 403.

```
from urllib.request import Request, urlopen
from bs4 import BeautifulSoup


url = "  https://pt.wikipedia.org/wiki/Fernanda_Torres"

# Adiciona um cabeçalho User-Agent
req = Request(
    url,
    headers={"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
                           "AppleWebKit/537.36 (KHTML, like Gecko) "
                           "Chrome/141.0.0.0 Safari/537.36"}
)

html = urlopen(req)
sopa = BeautifulSoup(html.read(), "html.parser")

links = sopa.find_all("a")
```

```
for link in links:
    href = link.get("href")
    print(href)
```

Contudo, ao observar o resultado obtido, podemos ver que há muitos links que não são desejados. Apresento abaixo, parte do resultado de nossa obtenção dos links dentro da página sobre Fernanda Torres. Veja que muitos links encontrados não se referem a artigos da Wikipédia.

```
#bodyContent
/wiki/Wikip%C3%A9dia:P%C3%A1gina_principal
/wiki/Portal:Conte%C3%BAdo_destacado
/wiki/Portal:Eventos_atuais
/wiki/Wikip%C3%A9dia:Esplanada
/wiki/Especial:Aleat%C3%B3ria
/wiki/Portal:%C3%8Dndice
/wiki/Especial:P%C3%A1ginas_especiais
/wiki/Wikip%C3%A9dia:Informe_um_erro
/wiki/Wikip%C3%A9dia:Boas-vindas
/wiki/Ajuda:P%C3%A1gina_principal
/wiki/Ajuda:P%C3%A1gina_de_testes
/wiki/Wikip%C3%A9dia:Portal_comunit%C3%A1rio
/wiki/Especial:Mudan%C3%A7as_recentes
/wiki/Wikip%C3%A9dia:Manuten%C3%A7%C3%A3o
/wiki/Ajuda:Guia_de_edi%C3%A7%C3%A3o/Como_come%C3%A7ar_uma_p%C3%A1gina
/wiki/Especial:P%C3%A1ginas_novas
/wiki/Wikip%C3%A9dia:Contato
/wiki/Wikip%C3%A9dia:P%C3%A1gina_principal
```

O que acontece é que a Wikipédia possui muitos links internos que não levam a outras páginas de artigos, mas apenas a uma posição dentro da mesma página (links como esses começam com #). Outros links referem-se apenas a cabeçalhos, rodapés e barras laterais da mesma página. Para obter apenas links de artigos úteis, que levam a outras

páginas de artigos (por exemplo, páginas sobre filmes ou sobre outros atores) precisaremos utilizar expressões regulares. Se você inspecionar a página em português da Wikipédia sobre Fernanda Torres, verá que todos os links que levam a artigos estão contidos dentro de uma tag **div** com o atributo **id** com o valor **mw-content-text**. Finalmente, outra característica das páginas de artigos é que elas não contêm em seu endereço o símbolo de dois pontos. Então, agora, nós vamos criar um padrão que estabeleça que o link desejado deva começar com /wiki/ e que não tenha : ou #. Veja como ficaria o script atualizado. Observe que dentro de colchetes, vamos adicionar elementos que não devem ter no padrão desejado. Por fim, usamos o cifrão ($) para dizer que o conteúdo do link termina logo após esse conteúdo curto e que não tem **:** e nem **#**. Quando você vir os resultados entenderá que não há nada mais além da estrutura **/wiki/nome_da_página**.

```
from urllib.request import Request, urlopen
from bs4 import BeautifulSoup
import re

url = "https://pt.wikipedia.org/wiki/Fernanda_Torres"
req = Request(url, headers={"User-Agent": "Mozilla/5.0"})
sopa = BeautifulSoup(urlopen(req).read(), "html.parser")

# Corpo do artigo (exclui menus e rodapés)
link = sopa.find("div", {"id": "mw-content-text"})

# Apenas links de artigos
padrao = re.compile(r"^/wiki/[^:#]*$")
for a in link.find_all("a", href=padrao):
    print(a["href"])
```

Veja abaixo parte dos resultados. Ao executar este código, veremos uma lista de todas as URLs de artigos para as quais o artigo da Wikipédia sobre Fernanda Torres aponta.

```
/wiki/Festival_Internacional_de_Cinema_de_Toronto
```

/wiki/Pseud%C3%B3nimo

/wiki/1965

/wiki/Rio_de_Janeiro

/wiki/Guanabara

/wiki/Brasil

/wiki/Brasileiros

/wiki/Fernanda_Montenegro

/wiki/Fernando_Torres_(ator)

/wiki/Pedro_Bial

/wiki/Gerald_Thomas_(diretor_de_teatro)

/wiki/Andrucha_Waddington

/wiki/Ator

/wiki/Escritor

/wiki/Roteirista

Apesar de ser um exercício interessante, podemos ampliá-lo para que ele seja algo mais generalizável. Podemos criar uma função getLinks que receba um URL de um artigo da Wikipédia no formato **/wiki/Nome_do_Artigo** e retorne uma lista com os URLs de todos os artigos associados no mesmo formato. Podemos, então, ter uma função principal que chame getLinks com um artigo inicial, escolha um link de artigo aleatório na lista obtida e chame getLinks novamente, até que o programa seja interrompido ou nenhum link de artigo seja encontrado na nova página. O novo código seria assim:

```
from urllib.request import Request, urlopen
from bs4 import BeautifulSoup
import re
import time
import random


random.seed(time.time())


UA = {"User-Agent": "Mozilla/5.0"}
```

```
def getLinks(url_do_artigo):
    url_completa = "https://pt.wikipedia.org" + url_do_artigo
    req = Request(url_completa, headers=UA)
    html = urlopen(req).read()
    sopa = BeautifulSoup(html, "html.parser")

    return sopa.find("div", {"id": "mw-content-text"}
                ).find_all("a", href=re.compile(r"^/wiki/[^:#]*$"))

links = getLinks("/wiki/Fernanda_Torres")
while len(links) > 0:
    novoArtigo = links[random.randint(0, len(links) - 1)].attrs["href"]
    print(novoArtigo)
    links = getLinks(novoArtigo)
```

A primeira coisa que fizemos foi usar um gerador de números aleatórios. Com ele, o programa pode escolher um link diferente a cada vez que roda, tornando o rastreamento mais dinâmico. Essa escolha aleatória acontece dentro de um loop, e faz o programa seguir de uma página para outra, como se estivesse passeando pela Wikipédia. O comando **seed** define o ponto de partida do gerador aleatório, e no nosso caso foi iniciado com o horário atual. Isso serve para garantir que os números sorteados sejam diferentes a cada execução. Na prática, essa linha é opcional, porque o Python já faz essa inicialização automaticamente. Em seguida, o programa define a função getLinks, que recebe o URL de um artigo no formato **/wiki/nome_do_artigo**, acrescenta o nome do domínio da Wikipédia (o início do endereço do site) como prefixo e obtém o objeto sopa para o HTML desse domínio. Depois da função que obtém os links de uma página, usamos um **loop while**, palavra que, em inglês, significa enquanto. Com ele, dizemos ao programa: Enquanto ainda existirem links na página, continue sorteando outro e siga para ele. Assim, o script navega de forma contínua e automática, encontrando novas páginas a partir dos links que encontra.

Como último exercício, vamos ampliar esta atividade. Agora, além de navegar por links de artigos da Wikipédia, vamos tentar obter os títulos dos artigos, o primeiro

parágrafo do conteúdo e o link para editar a página (se estiver disponível). Como sempre, antes de um exercício como este, devemos observar várias páginas do tipo que desejamos, inspecioná-las e verificar suas estruturas de HTML. Na Wikipédia, todos os títulos das páginas estão contidos dentro de tags **h1**, e essas são as únicas tag **h1** na página. Conforme mencionamos anteriormente, todo o texto do corpo em páginas em português está contido dentro de uma tag **div** com o atributo **id** com o valor **mw-content-text**. Já os links de edição aparecem em páginas de artigos e estão contidos dentro de uma tag **li** com o atributo **id** com o valor **ca-edit** ou **ca-viewsource**.

Há algumas diferenças nesse script um pouco mais avançado que você precisa estar atento. Inicialmente, nós vamos criar um **set** vazio chamado pages. Em Python, **set** é um conjunto de itens não ordenados e únicos, ou seja, sem repetição. Depois, estabelecemos um limite de páginas que vamos visitar, caso contrário seu script pode continuar executando por um tempo muito longo. Dentro da função estabelecemos que se o número de páginas (o número de elementos do set pages) é maior do que o limite estabelecido, o programa parará de funcionar.

Ainda dentro da função, depois de acessarmos uma página da Wikipédia, verificamos se existe um título com a tag **h1**. Caso exista, obtemos seu conteúdo sem tags filhas e sem espaços extras que possam existir. Em seguida, vamos ao conteúdo textual da página e verificamos se existe um parágrafo. Caso exista, imprimimos o início do primeiro parágrafo. Depois disso, verificamos se existem links de edição, e caso existam, imprimimos estes links. Após todo esse processo, a busca segue para outro artigo conforme vimos no script anterior. Observe também que usamos o método **time.sleep(0.5)**. A função sleep() do módulo time faz o programa "pausar" por um tempo, medido em segundos. O objetivo é evitar sobrecarga do servidor da Wikipédia. Vejamos o script completo abaixo:

```
from urllib.request import Request, urlopen
from bs4 import BeautifulSoup
import re
import time

UA = {"User-Agent": "Mozilla/5.0"}
pages = set()
```

```
LIMITE = 10  # número máximo de páginas que o crawler visitará

def getLinks(url_da_pagina):
    if len(pages) >= LIMITE:
        print("\nLimite de páginas atingido. Encerrando o rastreamento.")
        return

    url_completa = "https://pt.wikipedia.org" + url_da_pagina
    req = Request(url_completa, headers=UA)
    html = urlopen(req).read()
    sopa = BeautifulSoup(html, "html.parser")

    # --- título ---
    h1 = sopa.find("h1")
    if h1:
        print("\nTÍTULO:", h1.get_text(strip=True))

    # --- 1º parágrafo do conteúdo principal ---
    content = sopa.find("div", id="mw-content-text")
    primeiro_p = content.find("p") if content else None
    if primeiro_p:
        print("1º PARÁGRAFO:", primeiro_p.get_text(strip=True)[:200], "...")

    # --- link de edição (agora com a tag explícita <li>) ---
    ca_edit = sopa.find("li", id="ca-edit") or sopa.find("li", id="ca-viewsource")
    if ca_edit:
        a = ca_edit.find("a")
        if a and a.get("href"):
            print("LINK DE EDIÇÃO:", a["href"])
    else:
```

```
        print("Não há link de edição nesta página.")

    # --- segue para novos links (apenas /wiki/...) ---
    for link in sopa.find_all("a", href=re.compile(r"^/wiki/[^:#]*$")):
        href = link.get("href")
        if href and href not in pages:
            pages.add(href)
            print("-" * 40)
            print("Nova página:", href)
            time.sleep(0.5)
            getLinks(href)

# Começa em um artigo específico
getLinks("/wiki/Charles_Darwin")
```

Veja abaixo parte dos resultados obtidos com uma busca inicial na página da Wikipédia sobre Charles Darwin.

TÍTULO: Charles Darwin

1º PARÁGRAFO: Charles Robert DarwinFRSFGRSFLSFLZ[2](pronúncia em inglês:['dɑːrwɪn];[3]ocasionalmente aportuguesado emCarlos Darwin.[4]Shrewsbury,12 de fevereirode1809–Downe,19 de abrilde1882) foi umnaturalista,geól ...

LINK DE EDIÇÃO: /w/index.php?title=Charles_Darwin&action=edit

---

Nova página: /wiki/Charles_Darwin

TÍTULO: Charles Darwin

1º PARÁGRAFO: Charles Robert DarwinFRSFGRSFLSFLZ[2](pronúncia em inglês:['dɑːrwɪn];[3]ocasionalmente aportuguesado emCarlos Darwin.[4]Shrewsbury,12 de fevereirode1809–Downe,19 de abrilde1882) foi umnaturalista,geól ...

LINK DE EDIÇÃO: /w/index.php?title=Charles_Darwin&action=edit

---

Nova página: /wiki/Darwin_(desambigua%C3%A7%C3%A3o)

TÍTULO: Darwin (desambiguação)

1º PARÁGRAFO: Darwinpode referir-se a: ...

LINK DE EDIÇÃO:
/w/index.php?title=Darwin_(desambigua%C3%A7%C3%A3o)&action=edit

---

Nova página: /wiki/Darwin_(Austr%C3%A1lia)

Espero que este e-book tenha sido uma jornada de aprendizado e um despertar de curiosidade para todos os leitores. Seu puder ter tornado sua trajetória de aprendizado um pouco menos difícil, sinto que já terei cumprido meu objetivo. Web Scraping é um tema bastante dinâmico e interessante, considerando suas inúmeras aplicabilidades. Assim, este e-book é apenas um primeiro passo de um caminho desafiador, mas de muitas descobertas.

Referências

Garay, Federico. Python TOTAL - Programador Avanzado en 16 días. Udemy, 2023. Disponível em: https://www.udemy.com/course/python-total/?srsltid=AfmBOoqm3n4bADW4jBsJIdOG1mLFUpzzn-mvtjxGGRXQSAdo86QbUROf&couponCode=KEEPLEARNING. Acesso em: 17 out. 2025.

Mitchell, R. 2024. Web Scraping com Python: Coletando dados da web moderna. O'Reilly. 3 ed. Novated Editora LTDA.

W3Schools. *Python RegEx*. Disponível em: https://www.w3schools.com/python/python_regex.asp w3schools.com Acesso em: 16 de outubro de 2025.