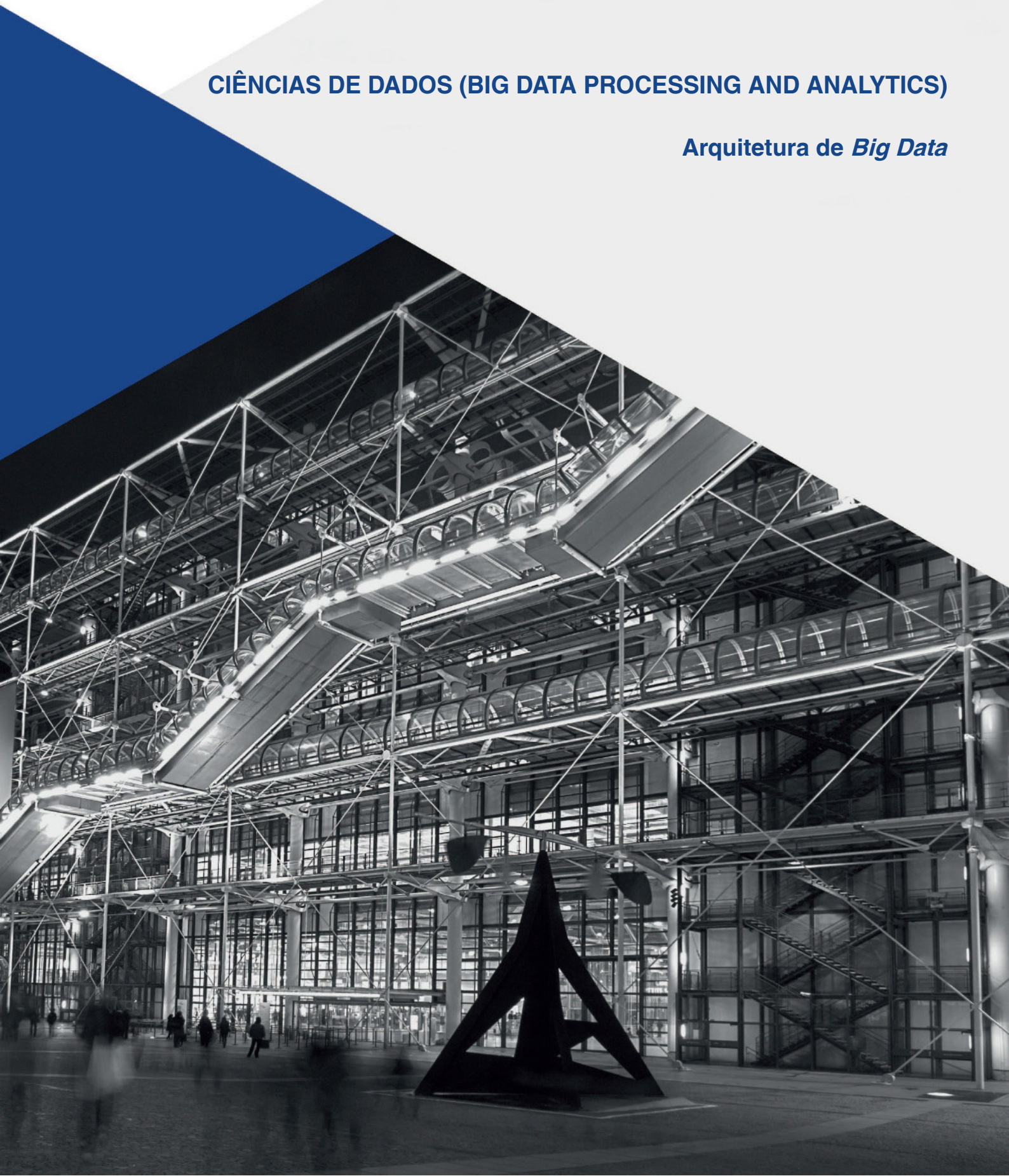
Universidade Presbiteriana Mackenzie

CIÊNCIAS DE DADOS (BIG DATA PROCESSING AND ANALYTICS)

Arquitetura de *Big Data*



Universidade Presbiteriana Mackenzie

Modalidade a distância

RECUPERAÇÃO DA INFORMAÇÃO NA WEB E EM REDES SOCIAIS

Trilha 3 – Recuperação de informação por raspagem: robôs

Professor: Luciano Moreira Camilo e Silva

Sumário

1. Introdução à Trilha

Na trilha anterior foi ensinado como um documento na web é tipicamente codificado e como podemos utilizar a ferramenta Beautiful Soup do Python para conseguir raspar esses dados e transformá-los em um formato mais adequado para o uso que se busca.

Seguindo com o aprendizado, esta trilha ajudará a pensar em como estruturar melhor suas ideias e programas, de forma a montar um sistema de raspagem de tela mais complexo, envolvendo múltiplos documentos. Algumas vezes envolvendo, inclusive, múltiplos endereços e estruturações de documentos diferentes.

Além disso, você já se perguntou como alguns sites que coletam informações na própria internet fazem para estar sempre atualizados?

Na primeira trilha, foi explicado como um sistema de recuperação de informação funciona, como ele indexa os documentos para conseguir responder às consultas dos usuários. Mas como o Google, por exemplo, consegue saber quais sites existem e está sempre descobrindo sites novos?

E não é apenas o Google que depende de coleta de informações. Sites de comparação de preços de hotéis, de passagens áreas ou de qualquer nicho dependem de estar sempre em buscas dos novos valores inclusos nos sites originais. Recrutadores raspam milhares de currículos diariamente a partir de palavras-chave.

Todos esses sistemas são baseados em robôs de captura de informação chamados de *web crawlers*.[1]

Ao final desta terceira trilha, você terá conhecimento suficiente para compreender e construir seus próprios robôs de coleta de informação baseados em documentos HTML.

Será apresentado um breve histórico das ferramentas, ampliando tanto a capacidade de inferência quanto o leque ferramental para uso diário. A biblioteca *Scrapy* do *Python* será utilizada nesta trilha.

Como exercício de fixação, criaremos um simples robô de captura de textos que coletará diversos textos de um famoso colunista brasileiro.

1 "Outros termos para rastreadores da rede são indexadores automáticos, robôs, aranhas da rede, robô da rede ou escutador da rede. (Em inglês: bots, web spiders, web robot e web scutter)." (Rastreador web, Wikipédia).

2. Web Crawlers

Robôs de catalogação de documentos (robôs de busca) estão presentes na internet desde o surgimento da web. Matthew Gray (http://www.mit.edu/~mkgray/bio.html) criou o primeiro sistema que buscava indexar todas as páginas da web, por meio de um script feito em ***pearl***.

Tabela 1 – Crescimento da Web, de acordo com o índice Wanderer de Matthey Gray

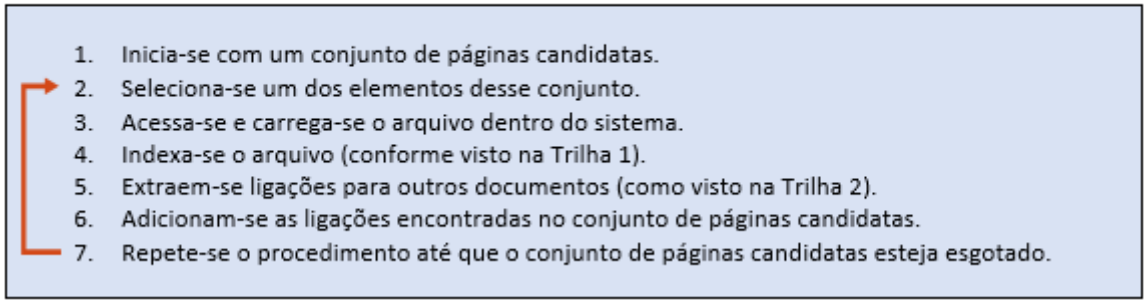
Mês	# de Web sites	% .com sites	Hosts per Web server
jun/93	130	1.5	13,000
dez/93	623	4.6	3,475
jun/94	2,738	13.5	1,095
dez/94	10,022	18.3	451
jun/95	23,5	31.3	270
jan/96	100	50.0	94
jun/96	230,000 (est)	68.0	41
jan/97	650,000 (est)	62.6	NA

Ao longo do tempo, diversos outros sistemas de indexação foram criados, mas pouco se sabe sobre eles. Como dizem Girardi; Ricca; Tonella (2006)

> *Mesmo que muitos web crawlers comerciais e de pesquisa estejam disponíveis na internet, há poucos trabalhos na literatura descrevendo os recursos e as questões de design dessas importantes ferramentas.*

Em seu trabalho, os autores compararam alguns dos robôs disponíveis para uso. De forma concisa, o que se percebe no trabalho deles e em diversas outras fontes acerca do assunto, como Najork (2009), Hsieh; Gribble; Levy (2010), Bahrami; Singhal; Zhuang (2015), é a descrição de um procedimento simples.

Figura 1 – Etapas de um robô de catalogação (robô de busca)



Apesar de ser simples de entender, o algoritmo traz uma série de desafios.

Alguns dos autores desses documentos podem não querer que eles sejam indexados, servidores podem sofrer com custos extras acessando diariamente seus documentos, sem que gerem nenhum valor a ele, e alguns servidores podem usar técnicas para banir robôs que abusam da velocidade de captura dessas informações, prejudicando a usabilidade de seus clientes e usuários.

Adicionalmente, como pontua Girardi; Ricca; Tonella (2006), nem todas as ligações estão no padrão HTML. A utilização de cookies e a autenticação, entre outros artifícios, podem ser utilizados pelos servidores para proteger seus dados e de seus usuários. Ninguém além do próprio Google deveria ver seus e-mails, certo?

Para pequenos projetos, muitos desses fatores podem ser ignorados. Porém, mesmo pequenos sistemas para coleta de informação precisam utilizar meios mais sofisticados para conseguir raspar as telas, o que vai além do que o Beautiful Soup, estudado na trilha passada, consegue fazer.

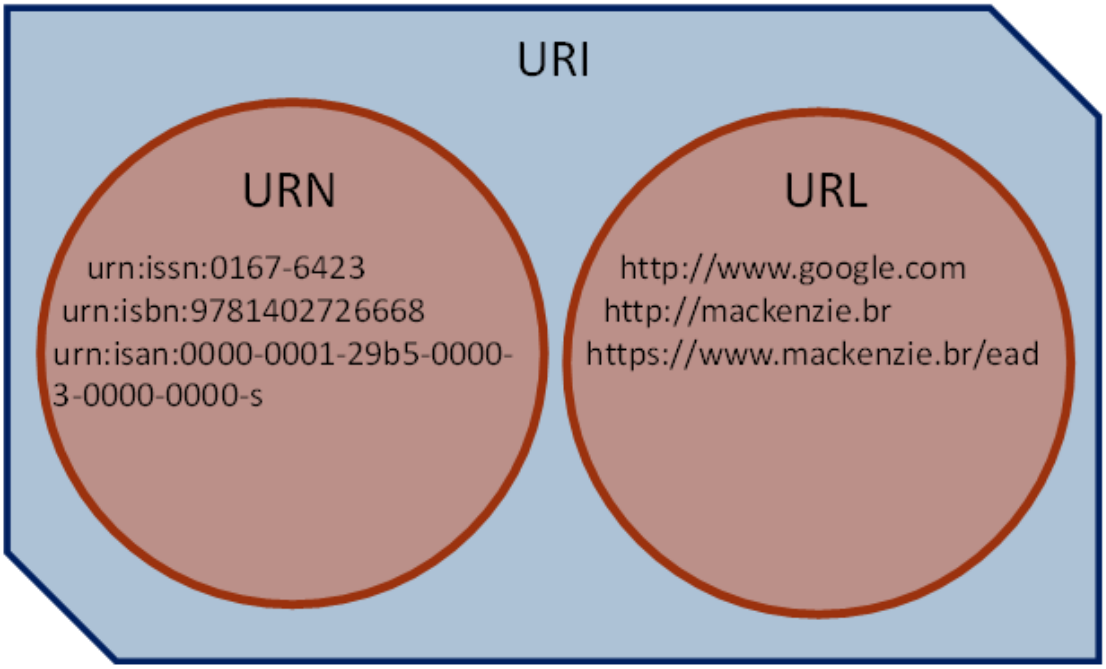
Para um uso mais complexo, como navegar por diversos web sites, podemos usar a biblioteca Python Scrapy (vide item 3). Mas antes, veremos dois itens importantes para raspagem de telas e construção de robôs.

2.1. URI – Uniform Resource Identifies

Uma padronização criada na internet para a nomenclatura ou localização de documentos é o Identificador de Recursos Uniformes, ou Uniform Resource Identifiers – URI, em inglês.

URI foi uma expansão dos trabalhos iniciais de Tim Bernes-lee no uso de Uniform Resource Locators (URL) e foi convertido na RFC 3986 (BARNES-LEE; MASINTER; FIELDING, 2005).

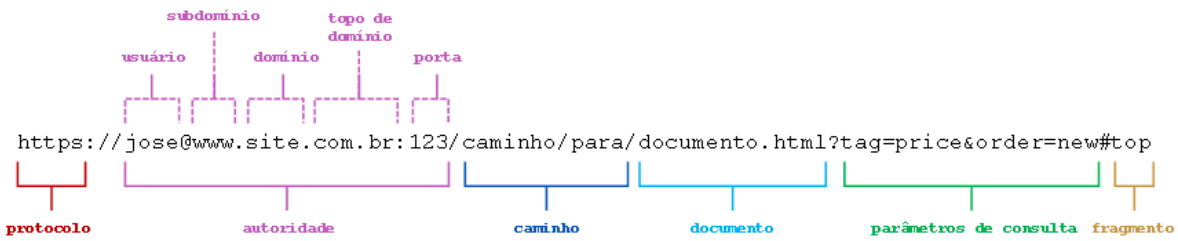
Figura 2 – Separação de URN e URL dentro do URI



Os URIs vieram complementar a ideia e versatilidade dos URLs e expandir para diversos usos. Hoje, você consegue usar URIs, em sua especificação *Uniform Resource Locators (URN)*, para identificar unicamente livros (urn:isbn:0-486-27557-4), filmes (urn:isan:0000-0000-2CEA-0000-1-0000-0000-Y), RFCs (urn:ietf:rfc:3986) entre tantos outros itens identificáveis.

Para raspagem de documentos web, entretanto, o que nos interessa é o URL. O URL é formado conforme mostra a Figura 3.

Figura 3 – Componentes de um URL



Fonte: Baseada em wikipedia.org.

- **Protocolo:** a web é apenas um pedaço da internet que trafega sob o protocolo aplicacional de transferência de hipertexto. Eles são identificados pelos acrônimos

http e https (quando trafegado de forma encriptada). Para esse módulo de raspagem de informações na web é o suficiente. Outros protocolos que utilizam de URL para identificar o destino são **ftp** e **sftp**, **news**, **mailto**, **tel**, **sms** ou mesmo específicos de algumas aplicações como **Skype** e **Spotify**.

• **Usuário:** campo opcional em uma URL. Tem como função passar uma identificação simples para o servidor, mais comum no protocolo (s)ftp. Pode ser combinado com a senha, no formato usuário:senha. Pode ser omitido no caso de domínios públicos, o mais comum na web.

• **Subdomínio, domínio e topo de domínio:** quando se registra um domínio, no caso do Brasil no site registro.br, escolhe-o concomitantemente com o nível superior (ou topo de domínio) do site. Atualmente, o registro.br possui 143 categorias de domínio diferentes disponíveis para compras (registro.br, 2021). O subdomínio pode ser combinado pelo desenvolver do website de forma a organizar melhor seu conteúdo, seja para segregar seções do site (esportes, entretenimento, restrito, público etc.), seja para separar projetos completamente diferentes (projeto1.site.com.br e projeto2.site.com.br).

• **Porta:** último componente da autoridade da URL. Quando utilizada, os valores padrões (80 para http e 443 para https) podem ser omitidos. Nos demais casos, deve-se informar a porta TCP/IP que responderá pelo pedido.

• **Caminho:** inicialmente, foi pensado para identificar o caminho até o arquivo no servidor. Se mostrou uma falha de segurança ao expor a estrutura de diretórios do servidor (Open Web Application Security Project, 2020). Apesar de continuar sendo usado com esse propósito por muitos sites, o caminho de uma URL hoje em dia pode ter comportamento de parâmetros de consulta tratado no servidor.

• **Documento:** identificação do documento que será acessado. Muitos servidores possuem um valor padrão quando este valor está omitido da URL, sendo o mais comum index.html.

• **Parâmetros de consulta:** iniciado a partir do ponto de interrogação, consistem em pares parâmetro=valor e podem ser concatenados diversos pares de parâmetro e valor, quando intercalados pelo caractere &. Sua função é passar informações ao servidor que será utilizado para melhor renderizar a página, como a ordenação de uma lista ou código de um produto que será listado. Atualmente, muitos desenvolvedores mascaram os parâmetros de consulta, incluindo ele como caminho e tratando os valores no servidor.

• **Fragmento:** iniciado a partir do símbolo #, esse componente identifica, dentro do documento, qual local se iniciar. Geralmente, não é tratado no servidor.

Uma forma de organizar sites, usada principalmente por aqueles oriundos de sistema de gestão de conteúdo, como blogs e sites de notícia, é utilizar caminhos personalizados.

Outra observação que o desenvolvedor de aplicativos de raspagem de tela precisa estudar é quanto aos parâmetros de ***query***. Esses parâmetros, muitas vezes, indicam para as aplicações web[1] o que deve ser filtrado. Aprendendo a lógica, é possível solicitar diretamente os recursos junto ao servidor e iterar, obtendo todos os valores desejados, como valores de uma ação na bolsa de valores, por exemplo.

2.2. HTTP Post

O protocolo de comunicação HTTP é a base da web e pode ser conhecido em detalhes no documento de quase 100 páginas da RFC 7540 (BELSHE; PEON; THOMSON, 2005).

Para a finalidade de raspagem de tela, é preciso compreender o básico do HTTP e seu método POST.

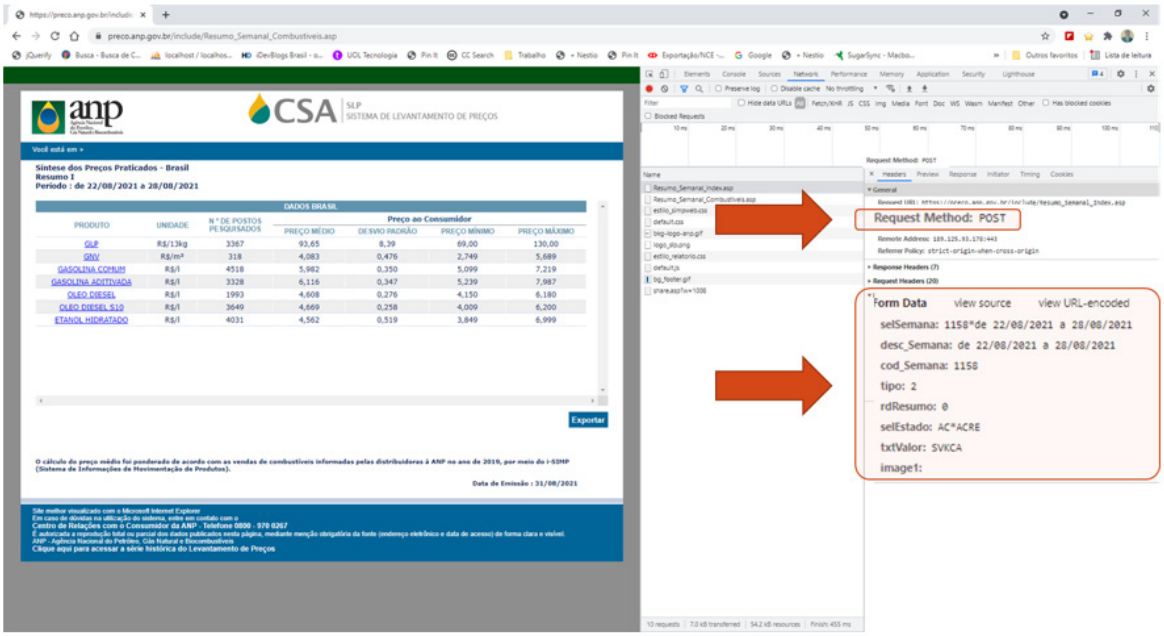
Quando se digita um URL no navegador, ele envia a requisição para o servidor por meio do método GET. O método GET foi projetado para consumir informações, apesar do que vimos no item anterior a respeito de parâmetros de query, os quais podem ser enviados para processamento do servidor.

Quando se trabalha com processamento de dados no servidor, o melhor padrão arquitetural é enviar essas variáveis por meio de um cabeçalho de requisição HTTP POST.

No processo de raspagem de tela, ao se identificar um padrão de envio dessas informações, pode-se utilizá-las para construir as requisições ao site. Muitas vezes, esses dados são procedentes de formulários (etiquetas <form> e <input>).

1 "Uma aplicação web também é definida em tudo que se é processado em algum servidor, exemplo: quando você entra em um e-commerce a página que você acessa antes de vir até seu navegador é processada em um computador ligado a internet que retorna o processamento das regras de negócio nele contido. Por isso se chama aplicação e não simplesmente site web ou um browser para permitir cookies de terceiros." (Aplicação web, Wikipédia).

Figura 4 – Site da ANP trabalhando com o método HTTP POST



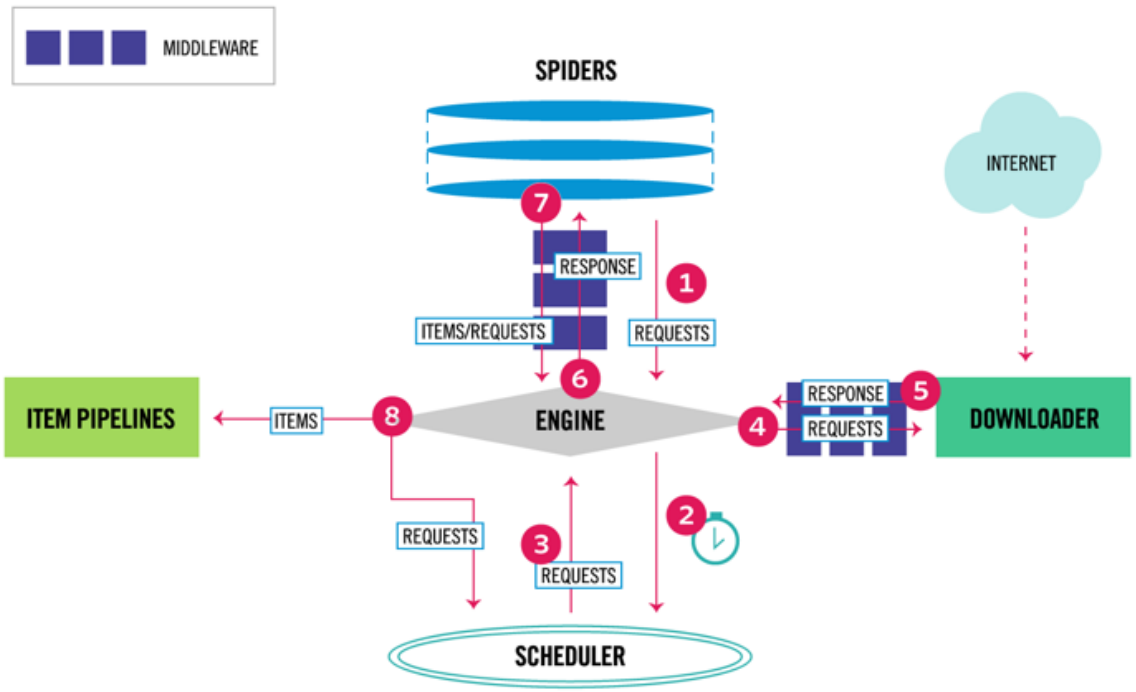
O envio desses parâmetros pelos robôs é tão simples quanto no padrão GET. Apenas é preciso, no momento do envio, configurar a requisição para usar o método em questão.

3. A biblioteca Python Scrapy

Diferentemente de uma raspagem de tela simples, construir um robô que busque continuamente e em diversos endereços conteúdo para ser raspado é uma atividade mais complexa e envolve uma série de funções (vide Figura 1).

O Scrapy é um *framework*[1] aplicacional que facilita a vida do desenvolvedor, pois ele gera automaticamente o código base dos vários componentes necessários para o robô funcionar. E permite que o desenvolvedor expanda as funcionalidades, se necessário, por meio de códigos personalizados.

Figura 5 – Arquitetura do framework Scrapy



A Figura 5 mostra como o Scrapy organizou em componentes distintos cada uma das etapas apresentada na Figura 1. A vantagem da componentização é a especialização que pode ser dada em cada uma das etapas.

Por códigos adicionais àqueles gerados pelo *framework*, pode-se acrescentar tratamentos

1 "Um framework em desenvolvimento de software, é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. Um framework pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação. Ao contrário das bibliotecas, é o framework quem dita o fluxo de controle da aplicação, chamado de Inversão de Controle." (Framework, Wikipédia).

da raspagem (por exemplo, descartar a leitura de um produto em uma loja de comercio eletrônico, caso ele esteja nulo ou em valores fora do aceitável), atrasar os pedidos de requisição para evitar ser banido pelo servidor de destino ou fazer alguns tratamentos de dados para facilitar o uso posterior.

O principal componente que precisa ser escrito é o robô ou, como o framework prefere dizer, a aranha.

A classe possui algumas propriedades opcionais que podem ser utilizadas para customizar seu robô:

- **name:** o nome único do seu robô, dentro do projeto em questão. Como boa prática, se seu robô funciona apenas para um domínio, utilize-o: se robô interpreta apenas os dados de http://siteexemplo.com.br, dê o nome de br.com.siteexemplo.
- **allowed_domains:** uma lista com todos os domínios permitidos para a busca do seu robô. Utilize para restringir a busca caso ele encontre alguma ligação externa.
- **start_urls:** uma lista inicial de URLs para seu robô buscar.

Além dessas propriedades, existem os seguintes métodos padrão, chamados pelo framework:

- **start_requests():** o nome único do seu robô, dentro do projeto em questão. Como boa prática, se seu robô funciona apenas para um domínio, utilize-o: se robô interpreta apenas os dados de http://siteexemplo.com.br, dê o nome de br.com.siteexemplo.
- **parse(self, response, **kwargs):** esta classe DEVE ser implementada por você e é o coração do robô, a peça que faz a raspagem de tela. Adicionalmente, você pode criar interpretadores adicionais, com outros nomes, para avaliar documentos diferentes. O objeto "response" contém o documento que foi recebido do downloader (vide Figura 5).

```
1.  class MySpider(scrapy.Spider):
2.      name = 'exemplo.com.br'
3.      allowed_domains = ['exemplo.com.br']
4.
5.      def start_requests(self):
6.          yield scrapy.Request('http://www.exemplo.com.br/lista.html', self.parse)
7.
8.      def parse(self, response):
9.          for href in response.css('a::attr(href)').getall():
10.             yield scrapy.Request(response.urljoin(href), self.parse_text)
11.
12.     def parse_text(self, response):
13.         yield response.css('*::text')
```

O Scrapy vem nativamente com uma biblioteca para interpretar o HTML com algumas propriedades muito interessantes. Ela possui algumas propriedades como url para descrever a URL que está sendo processada, headers, um dicionário Python com todas as propriedades utilizadas no cabeçalho da chamada, status que permitem você identificar qual foi o retorno do HTTP STATUS CODE do servidor, além do dicionário meta que pode ser utilizado para passar variáveis do robô entre as chamadas do Scrapy.

Além disso, há a função response.urljoin(url) que converte ligações relativas em valores absolutos, baseadas na URL de requisição, e o método response.css() que lhe permite fazer consultas no HTML, utilizando seletores, como os vistos em *javascript* (ETEMAD et al., 2018). Além disso, o Scrapy possui dois operadores não padronizados, os quais, porém, ajudam muito no processo de raspagem de documentos na web.

```
# Seleciona todos os elementos da etiqueta p
response.css('p').getall()

# Seleciona todos os elementos div da classe produto
response.css('div.produto').getall()

# Seleciona o elemento de id título
response.css('#titulo').get()

# Seleciona todos os elementos das etiquetas que têm a propriedade href
# operador não padronizado
response.css('a::attr(href)').getall()

# Seleciona todos os textos do elemento span
# operador não padronizado
response.css('span::text').getall()
```

3.1. Dicas rápidas

Caso queira interagir com alguma página, da mesma forma que o Scrapy, você pode utilizar o comando de linha abaixo. É uma forma rápida e fácil de encontrar as etiquetas que deseja raspar.

```
scrapy shell <url>
```

Caso esteja acostumado com o Beautiful Soup (visto na trilha anterior), pode importá-lo junto ao seu projeto, passando o texto do response.

```
from bs4 import BeautifulSoup
...
def parse_text(self, response):
    soup = BeautifulSoup(response.text, 'html.parser')
    ...
```

Caso necessite enviar dados pelo método HTTP POST, como no exemplo da Figura 4,

deve usar o comando request.FormRequest. Outro uso comum é para se autenticar com um usuário e senha.

```
1.  class MySpider(scrapy.Spider):
2.      name = 'myspider'
3.
4.      def start_requests(self):
5.          return [scrapy.FormRequest("http://www.example.com/login",
6.                                     formdata={'user': 'john', 'pass': 'secret'},
7.                                     callback=self.logged_in)]
8.
9.      def logged_in(self, response):
10.         # seu código aqui
11.         pass
```

Pode evitar o uso da função start_requests(), caso haja uma lista inicial limitada de URLs e seja utilizada a função de interpretação parse(). Basta declarar a lista no vetor start_urls.

```
1.  class MySpider(scrapy.Spider):
2.      name = 'example.com'
3.      allowed_domains = ['example.com']
4.      start_urls = [
5.          'http://www.example.com/1.html',
6.          'http://www.example.com/2.html',
7.          'http://www.example.com/3.html',
8.      ]
9.
10.     def parse(self, response):
11.         self.logger.info(f"A página do endereco {response.url}foi capturada!")
```

Para extrair qualquer link do documento, utilize o extrator nativo do Scrapy.

```
1.      for link in self.link_extractor.extract_links(response):
2.          yield Request(link.url, callback=self.parse)
```

Salve suas raspagens em diversos formatos por meio da linha de comando. Acrescente o arquivo.formato durante a execução. Os formatos aceitos são CSV, JSON e JL (JSON Lines).

```
1.  scrapy crawl <spider name> -o arquivo.csv -t csv
```

4. Exercício prático

Este exercício tem como objetivo coletar dados do famoso colunista brasileiro Arnaldo Jabor. Esses textos serão utilizados novamente em um novo exercício nas trilhas posteriores.

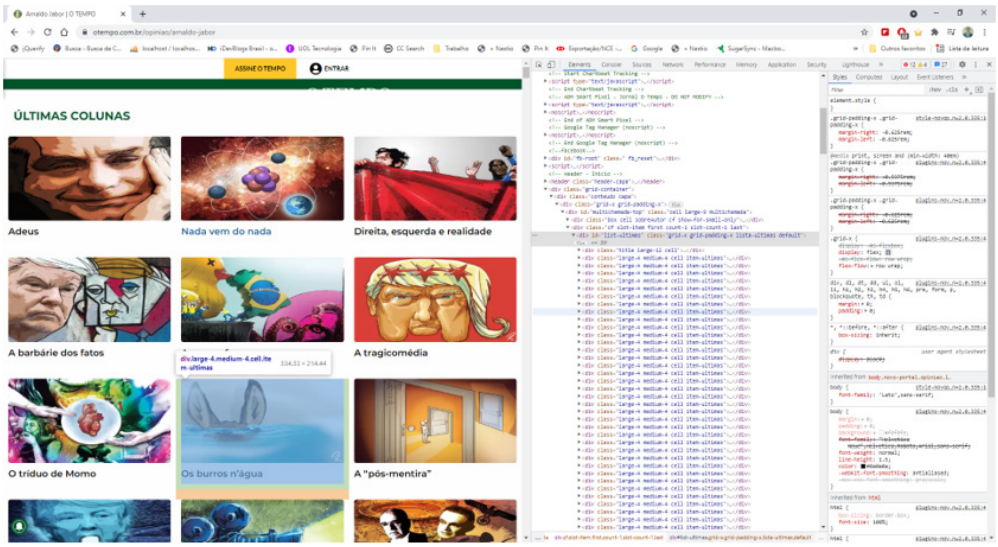
4.1. Conhecendo o conteúdo a ser raspado

Como em todo exercício de raspagem de tela, a primeira etapa é encontrar fontes confiáveis para nosso projeto. Arnaldo Jabor escreveu por quase dez anos e, para esse projeto, escolhemos como fonte o jornal *O Tempo* de Minas Gerais.

As colunas de Arnaldo Jabor ficam disponíveis na ligação https://www.otempo.com.br/opiniao/arnaldo-jabor e podem ser publicamente acessadas por todos.

Analisando o documento (vide Figura 1), percebemos que todas as colunas estão identificadas pela etiqueta ... . Dentro dessa etiqueta, encontra-se o link para as colunas na etiqueta <a>. Um cuidado que se deve observar é que existem três hiperlinks, logo, de forma a otimizar nosso robô, pegaremos apenas um deles.

Figura 6 – Inspeção no site do jornal O Tempo



A próxima etapa é identificar como o texto aparece no documento. Sem alongar muito nos detalhes do trabalho de inspeção, conclui-se que o texto está em etiquetas de parágrafo <p> dentro de uma etiqueta de divisão .

Agora é hora de codificar!

4.2. Executando o código Python

Para iniciar um projeto Scrapy, após instalado em sua máquina local, utilize a aplicação em comando de linha auxiliar para iniciar um diretório com todos os arquivos necessários para o desenvolvimento.[1]

```
scrapy startproject robo_aj
```

E, na sequência, execute:

```
cd robo_aj
scrapy startproject robo_aj
```

O comando criará uma estrutura de pastas, como mostrado abaixo. Para este exercício, não será necessário alterar nenhuma dessas configurações.

```
robo_aj/
  scrapy.cfg          # Arquivo de configuração principal
  robo_aj/
    __init__.py
    items.py          # Arquivo de modelagem de itens
    middlewares.py    # Arquivo de parametrização das requisições
    pipelines.py      # Arquivo de configurações do pipeline
    settings.py       # Arquivo de configurações do robô
    spiders/
      __init__.py
          spider_aj.py  # Arquivo que conterá seu robô
```

Abra o arquivo ./robô_aj/spiders/spider_aj.py para codificar o robô.

De forma a não ter que lidar com a complexidade de um site dinâmico,[2] navegue pelo site do jornal O Tempo até que todas as matérias estejam carregas e, então, salve-as no computador. O arquivo também estará disponível para os alunos.

Para identificar o caminho completo do arquivo, utilizaremos as bibliotecas nativas do

1 Você pode utilizar os aplicativos **PowerShell** no Windows. Caso esteja usando uma máquina Mac OS ou Linux, utilize o **Terminal**. Ou utilize o aplicativo de Shell de Linha de Comando favorito.

2 Caso você esteja interessado em lidar com automação de navegadores e sites dinâmicos, sugiro conhecer o **Selenium** no endereço https://www.selenium.dev e verificar como utilizá-lo em conjunto com o Python.

sistema operacional e outra que ajuda na composição da caminhos (os e pathlib).

Dado que a fonte de todas as ligações está em um único documento, pode-se usar a variável da classe start_urls. Será necessário, entretanto, montar duas funções de interpretação (*parsers*, em inglês).

A primeira função montará o link para os textos. Como visto no item 4.1, sabe-se que o link é encontrado no caminho response.css(“.item-ultimas”).css(“h2”).css(“a::attr(href)”). Uma vez iterado em todos esses valores, combinando-o com o restante do endereço, requisita-se a página do artigo que será decifrada pelo segundo interpretador.

Novamente, como visto no tópico 4.1, sabe-se que o texto está disponível no item response.css(‘div.texto-artigo p::text’). Será raspado o título do artigo e, como boa prática, um atributo para o autor do texto será criado. Isso facilitará juntar este artigo com outros quando necessário.

Abaixo, o código final.

```
1.  import scrapy
2.  import os
3.  import pathlib
4.  import csv
5.
6.  class SpiderAj2Spider(scrapy.Spider):
7.      name = "spider_aj"
8.
9.      # Abre o arquivo local, com todos os links
10.     start_urls = [f"{pathlib.Path(os.path.abspath('arnaldo_jabor.html')).as_uri()}"]
11.
12.     # Apaga o arquivo caso exista, para evitar sobreposição
13.     if os.path.exists("artigos_aj.csv"):
14.         os.remove("artigos_aj.csv")
15.
16.     # função que interpreta o documento que lista os artigos
17.     def parse(self, response):
18.         for link in response.css(".item-ultimas").css("h2").css("a::attr(href)").getall():
19.             text_page = f"https://www.otempo.com.br/{link}"
20.             yield scrapy.Request(text_page, callback=self.parse_text)
21.
22.     # função que interpreta os documentos com os textos
23.     def parse_text(self, response):
24.         content = ""
25.         for line in response.css('div.texto-artigo p::text').getall() :
26.             content = content + "".join(line) + "\n"
27.
28.         post = {
29.             'author': 'Arnaldo Jabor',
30.             'title': response.css('h1::text').get(),
31.             'content': content.encode('utf-8')
32.         }
33.
34.         with open('artigos_aj.csv', 'a', newline='', encoding="utf-8")  as output_file:
35.             dict_writer = csv.DictWriter(output_file, post.keys())
36.             dict_writer.writerows([post])
```

Execute o código para ver o arquivo artigos_aj.csv gerado em sua máquina.

```
scrapy runspider .\robo_aj\spiders\spider_aj.py
```

5. Síntese

Com este exercício encerra-se o conjunto introdutório do módulo de Recuperação da Informação na Web e em Redes Sociais.

O aprendizado até aqui permitiu que você compreendesse como os sistemas de recuperação da informação evoluíram desde os primórdios, ainda com máquinas mecânicas, as quais já utilizavam conceitos que são tão pertinentes ao mundo do sistema de informação, como índices invertidos e algoritmos de ranqueamento.

Passamos pelo surgimento da web e suas tecnologias correlatas, como o protocolo HTTP e a linguagem de marcação HTML. E aprendemos a raspar dados de uma sopa de etiquetas. Nesta trilha, especificamente, aprendemos a construir nosso próprio sistema de recuperação de informação.

Antes de exercitar e construir seu próprio robô, pense a respeito dessas questões:

- O que eu aprendi até esse momento? No que eu preciso me aprofundar?
- O que faz um robô de captura de informação? E como ele se diferencia de um sistema de busca?
- Quais sites ou aplicações que eu vejo no meu dia a dia trabalham com raspagem de informação?
- Onde eu poderia aplicar um novo robô de captura de documentos para resolver problemas do dia a dia da minha empresa ou da minha rotina?
- Quais oportunidades de mercado poderiam ser aproveitadas utilizando o que aprendi até o momento?

Pense nessas perguntas e faça exercícios para fixar tudo que aprendeu até aqui. Como diz o ditado popular, reiterado pelo estudo de Ericsson; Krampe; Tesch-Romer (1993):

> *A prática leva à perfeição.*

5. Referências

BAHRAMI, M.; SINGHAL, M.; ZHUANG, Z. A cloud-based web crawler architecture. In: 18th International Conference on Intelligence in Next Generation Networks, IEEE, p. 216-223, fev. 2015.

BELSHE, M.; PEON, R.; THOMSON, M. (eds). Hypertext Transfer Protocol Version 2 (HTTP/2). ***RFC Editor***, maio 2015. Disponível em: <doi:10.17487/RFC7540>. Acesso em: 15 out. 2021.

BERNERS-LEE, T.; MASINTER, R. T.; FIELDING, L. M. Uniform Resource Identifier (URI): Generic Syntax. ***RFC Editor***, jan. 2005. Disponível em: <doi:10.17487/RFC3986>. Acesso em: 15 out. 2021.

ERICSSON, K. A.; KRAMPE, R. T.; TESCH-ROMER, C. The Role of Deliberate Practice in the Acquisition of Expert Performance. ***Psychological Review***, v. 100, n. 3, p. 363, 1993.

GIRARDI, C.; RICCA, F.; TONELLA, P. Web crawlers compared. ***International Journal of Web Information Systems***, v. 2, n. 2, p. 85-94, 1º maio 2006. Disponível em: <doi:https://doi.org/10.1108/17440080680000104>. Acesso em: 15 maio. 2021.

HSIEH, J. M.; GRIBBLE, S. D.; LEVY, H. M. The Architecture and Implementation of an Extensible Web Crawler. ***NSDI***, p. 28-30, 2010.

NAJORK, M. Web Crawler Architecture. ***Microsoft Research***, 2009.

SCRAPY Developers. ***Scrapy Documentation***, Release 2.5.0, 6 out. 2021. Disponível em: <https://docs.scrapy.org/_/downloads/en/2.5/pdf/>. Acesso em: 15 maio 2021.