Web Scraping com Python

Sobre mim

```
{
  name: "Kaio Duarte",
  email: "duartedossantos@gmail.com",
  favoriteLanguage: "python",
  social: [
    "github.com/kaioduarte",
    "linkedin.com/kaiodduarte",
    "facebook.com/duartedossantos"
  ]
}
```

LET’S GET IT STARTED!

O que é web scraping?

É a prática de coletar dados através de qualquer meio que não seja uma API ou através de um humano usando um navegador.

É comumente feito através de um programa automatizado que envia requisições para servidores, solicitam dados e então analisam os dados para extrair as informações necessárias.

Quem usa isso? ¯_(ツ)_/¯

- Google
- Decolar.com
- Buscapé
- Escavador
- E muitas outras empresas...


trivago
trivago®
a sua busca
pelo hotel
ideal

Skills úteis para web scraping

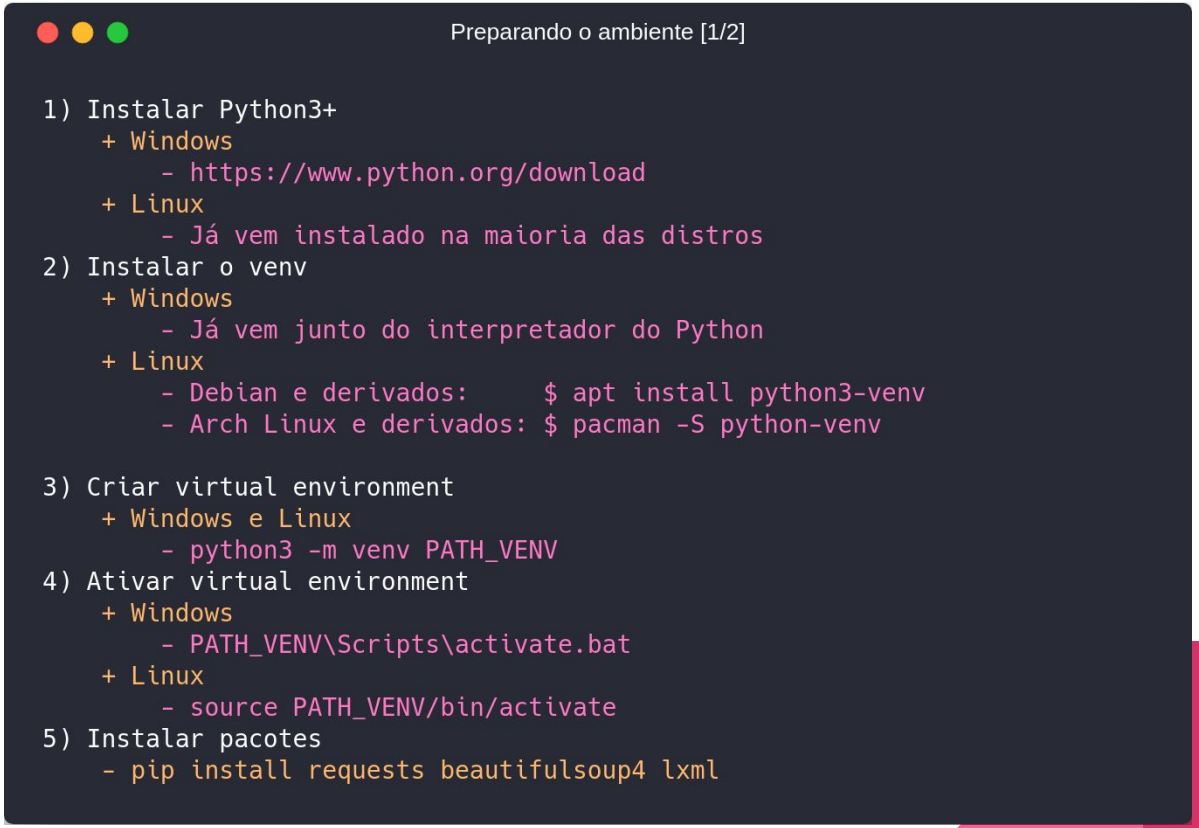
- HTML
- CSS
- Javascript
- Expressões regulares
- Compreender a criação de websites

Stack Python para web scraping

- **Requests**
- Requests-HTML
- **BeautifulSoup**
- Parsel
- **Selenium**
- Scrapy

Preparando o ambiente

Preparando o ambiente [1/2]

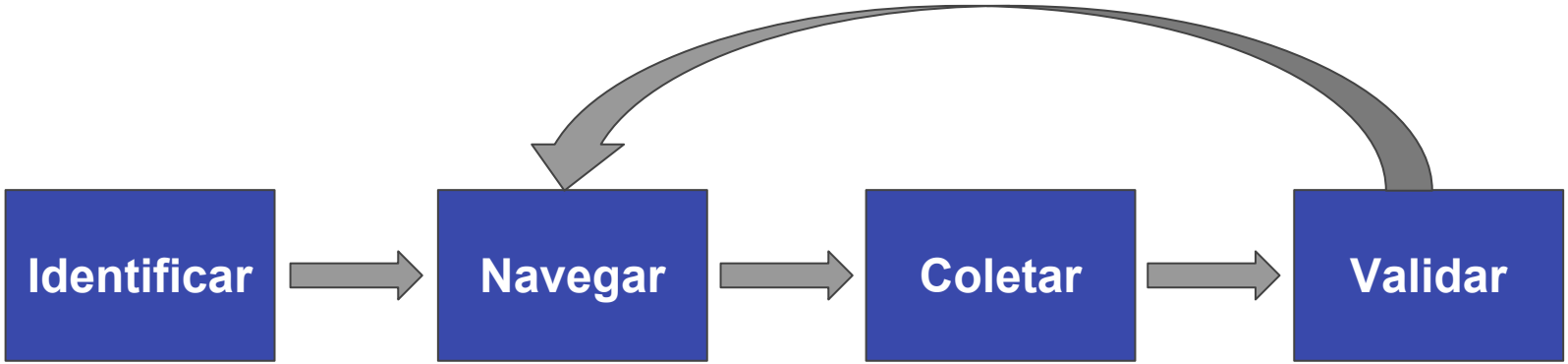


Preparando ambiente [2/2]

Preparando o ambiente [2/2]

```
1) Instalar o selenium
    - pip install selenium

2) Baixar o web driver do seu navegador
    + Chrome
        - http://chromedriver.chromium.org/downloads
    + Firefox
        - https://github.com/mozilla/geckodriver/releases

3) Descompactar o arquivo e mover para a pasta /bin do virtual
environment criado.
```

Pipeline web scraping



Identificar

Identificamos a informação que queremos coletar. Aqui precisamos entender bem qual é a estrutura das páginas que queremos raspar e traçar um plano para extrair tudo que precisamos.

Navegar

Agora precisamos entender de onde e como vem o dado que queremos extrair. Para isso vamos usar a ***Aba do desenvolvedor*** do navegador.

Coletar

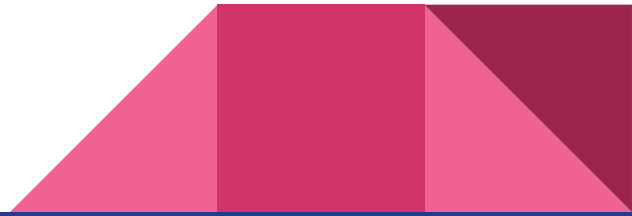
Extraí-se os dados identificados do HTML.

Validar

Se tivermos feito tudo certo até agora, validar os resultados será simples. Precisamos apenas reproduzir o procedimento descrito até agora para algumas outras páginas de modo verificar se estamos de fato extraindo corretamente tudo o que queremos.

Let’s dive into it!

- Requisições HTTP
- Navegar pelo DOM
  - BeautifulSoup
  - CSS Selectors



Requisições HTTP

Método	Semântica
GET	“Servidor, me dê o recurso *tralalá*”
POST	“Servidor, processa esses dados (cria um recurso)”
PUT	“Servidor, crie ou edite esse recurso”
DELETE	“Servidor, apague esse recurso”

Requisições HTTP

Família de retorno	Semântica
1xx	“Requisição recebida e está sendo processada”
2xx	“Requisição bem sucedida”
3xx	“Informa ação adicional a ser tomada para completar a requisição”
4xx	“Aviso de que a requisição não pôde ser atendida”
5xx	“Erro no servidor ao executar uma requisição válida”

Requisições HTTP (*requests*)

```python
import requests

url = 'https://jsonplaceholder.typicode.com/posts'

payload = {
    'title': 'foo',
    'body': 'bar',
    'userId': 1
}

requests.get(url)

requests.post(url, data=payload)
```

Requisições HTTP (*requests*)

```python
import requests

url = 'https://jsonplaceholder.typicode.com/posts/1'

payload = {
    'id': 1,
    'title': 'foo',
    'body': 'bar',
    'userId': 1
}

requests.delete(url)
requests.put(url, data=payload)
```

DOM (Document Object Model)

Documento representado por uma estrutura de árvore, onde cada nó é um objeto que representa uma parte do documento.

Exemplos: HTML, XHTML e XML.

BeautifulSoup

Biblioteca que auxilia na extração de dados de arquivos HTML e XML. Possui métodos *pythônicos* para navegação, busca e modificação do DOM.

Navegação pelo DOM (BeautifulSoup)

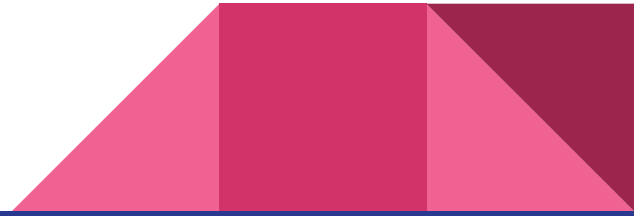
```
from bs4 import BeautifulSoup

# O parser default é o 'html.parser', mas existem outros
dom = BeautifulSoup(<string>, <parser>)

# Retorna o primeiro parágrafo do DOM
dom.find('p')

# Retorna todos os parágrafos do DOM
dom.find_all('p')

# Retorna todos os parágrafos com a classe 'xpto'
dom.find_all('p', class_='xpto')
dom.find_all('p', { 'class': 'xpto' })
```

Navegação pelo DOM (BeautifulSoup)

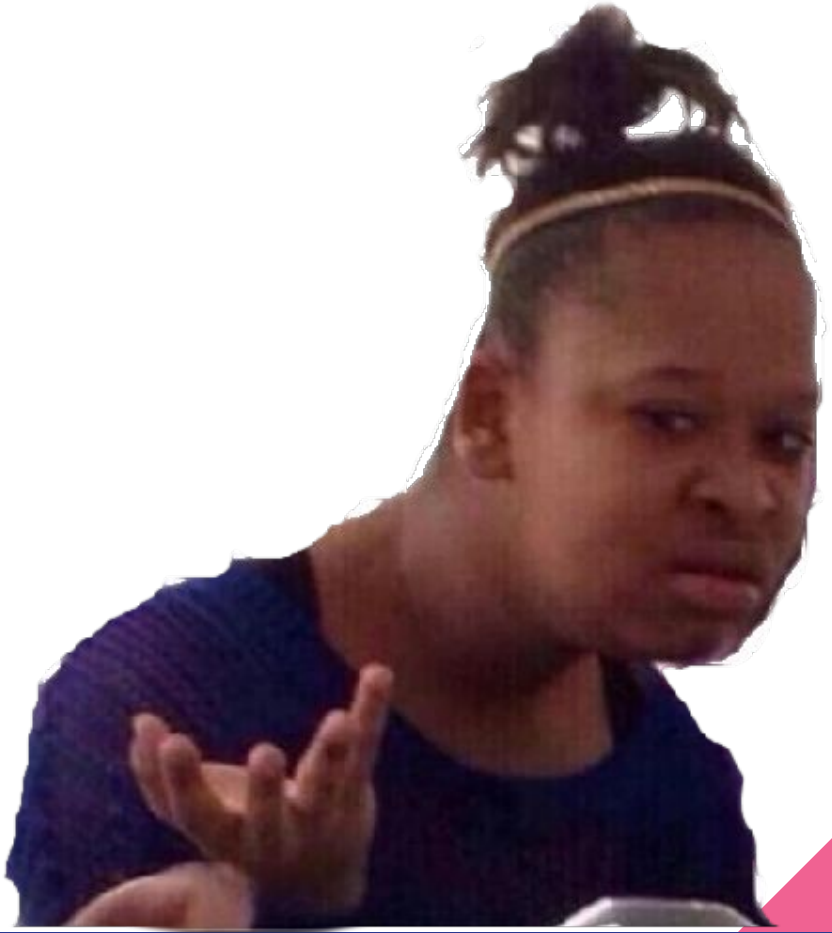
```
# Exibe o texto da 1ª tag 'p'
p = dom.find('p')
print(p.text)

# Exibe o atributo 'href' da 1ª tag 'a'
a = dom.find('a')
print(a['href'])

# Exibe os nós filhos da tag 'a'
print(a.children)

# Exibe o nó pai da tag 'a'
print(a.parent)
```

CSS Selectors

Em CSS, os seletores são padrões usados para selecionar um elemento desejado para se estilizar.



Tá, mas e aí?



CSS Selectors

Os seletores são muito úteis em web scraping, pois facilita a localização dos elementos no DOM.

 # TODO
Estudem XPath!

CSS Selectors - Sintaxe

Seletor	Exemplo	Descrição
#id	#product	Seleciona o elemento com o id=”product”
.classe	.product-item	Seleciona os elementos que tenham a class=”product-item”
elemento	p	Seleciona as tags <p>
[atributo=valor]	[rel=canonical]	Seleciona as tags que possuem o atributo rel=”canonical”

CSS Selectors - Sintaxe

Seletor	Exemplo	Descrição
elemento > elemento	div > p	Seleciona as tags <p> que são filhas de uma
elemento + elemento	div + p	Seleciona todas as tags <p> que estão localizadas imediatamente após tags
elemento elemento	div a	Seleciona as tags <a> que são descendentes de uma

CSS Selectors (BeautifulSoup)

CSS Selectors

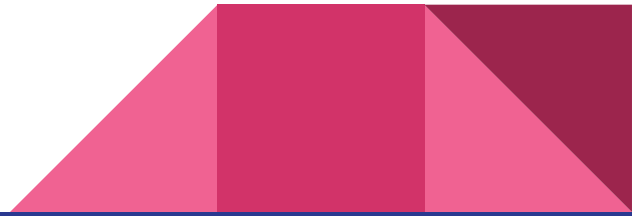
```
# Exibe a primeira tag que possui o atributo rel='canonical'
print(dom.select_one('[rel=canonical]'))

# Exibe todas as tags 'a' que possuem a classe 'product'
print(dom.select('a.product'))

# Os seletores podem ser combinados
dom.select('#product .product-item > a')
```

HANDS ON!

Web Scraping e automação com *Selenium*

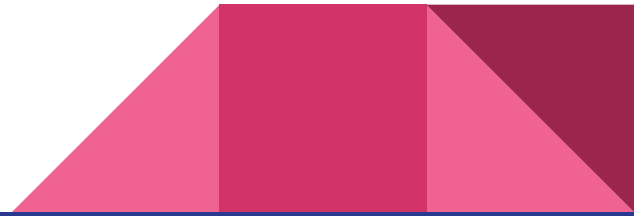
O que é o Selenium?

Um framework que permite realizar interações com o navegador web, como cliques, scroll, screenshot, etc.



Qual a utilidade disso em web scraping?

Algumas páginas renderizam seu conteúdo dinâmicamente com Javascript e as requisições para os servidores não serão suficientes. Como o Selenium simula um navegador, aí está a sua utilidade ;)



Métodos do Selenium

Métodos do Selenium

```
from selenium import webdriver

# Instancia um navegador (é necessário ter o
# navegador instalado e o seu webdriver)
driver = webdriver.Firefox()
driver = webdriver.Chrome()

url = 'https://www.google.com'

# Carrega uma página
driver.get(url)

# Localiza um elemento pelo seletor CSS
text_field = driver.find_element_by_css_selector('#lst-ib')

# Envia um texto para o text field
text_field.send_keys('olá mundo')

# Envia o formulário
text_field.submit()
```

Referências

- Inspirações:
  - curso-r.com/blog/2018-02-18-fluxo-scraping/
  - Live de Python
- Documentações:
  - crummy.com/software/BeautifulSoup/bs4/doc/
  - docs.python-requests.org/en/master/
  - selenium-python.readthedocs.io
- Indicações:
  - Live de Python: youtube.com/eduardomendes