



# PYTHON – AULA 5

## WEB SCRAPING

# AGENDA

## Aula 1

Introdução – conceitos, arquitetura cliente – servidor, Introdução a HTML, CSS e JavaScript, URLLIB e Requests.

## Aula 2

Introdução ao BeautifulSoup, Expressões Lambdas, Expressões regulares (REGEX)

## Aula 3

BeautifulSoup, List Comprehension, Revisitando NUMPY e PANDAS, Introdução ao Selenium

## Aula 4

Selenium 4, Scraping de Imagens, Trabalhando com inputs em pesquisas, Introdução ao Docker.

## Aula 5

Selenium 4, Banco de dados SQLite, projeto final, considerações finais

# WEB SCRAPING

SELENIUM 4

SQLITE

MINI PROJETO FINAL

CONSIDERAÇÕES FINAIS

## Web Scraping – Selenium



**Selenium**

- O Selenium é uma biblioteca de testes para manipulação de páginas dinâmicas.
- Automatiza browsers
- Permite simular um usuário real utilizando um navegador
- Mais indicado para sites que contêm muito código Javascript, JQuery, Angular, Vue e React.

## Web Scraping – Selenium 4



Selenium

- Com a chegada do Selenium 4, não precisamos mais gerenciar os drivers (versão do navegador)
- Tivemos também algumas mudanças em relação às classes e seus métodos.
- No núcleo do Selenium 4, está o WebDriver, uma interface para escrever conjuntos de instruções que podem ser executados de forma intercambiável em muitos navegadores.
- O Selenium WebDriver é uma recomendação do W3C. Uma API compacta orientada a objetos, que “dirige” o navegador de forma eficaz.



## Web Scraping – Selenium 4

### Classe By

Esta classe tem as estratégias de localização. Vamos utilizar um `find_element` genérico e passamos as estratégias.

#### selenium.webdriver.common.by

The By implementation.

##### Classes

`By` Set of supported locator strategies.

`class selenium.webdriver.common.by.By`

Set of supported locator strategies.

`CLASS_NAME = 'class name'`

`CSS_SELECTOR = 'css selector'`

`ID = 'id'`

`LINK_TEXT = 'link text'`

`NAME = 'name'`

`PARTIAL_LINK_TEXT = 'partial link text'`

`TAG_NAME = 'tag name'`

`XPATh = 'xpath'`

## Web Scraping – Selenium 4

### find\_element e find\_elements

- `find_element` devolve um elemento único
- Ex: `driver.find_element(By.CLASS_NAME, "value").text`
- `find_elements` devolve uma lista de elementos
- Ex: `driver.find_elements(By.CLASS_NAME, "value")[0].text`
- Estes métodos podem ser utilizados com os demais atributos da classe `By`.




- **Web Scraping – Selenium 4**
- **Baixando imagens**

- +
  - +
    - Vamos inspecionar a página IMDB no carousel de fotos do filme Dungeons & Dragons. Ao clicar sobre as imagens encontramos o container; podemos copiar o xPath.

Assistir a Dungeons & Dragons: Wrath of the Dragon God

**div.ipc-sub-grid.ipc-sub-grid--page-span-2.ipc-sub-grid--nowrap.ipc-shoveler\_grid** 363 × 145.5



```
</div>
...
<div class="ipc-sub-grid ipc-sub-grid--page-span-2 ipc-sub-grid--nowrap ipc-shoveler_grid" data-testid="shoveler-items-container">
  <div class="ipc-photo ipc-photo--base ipc-photo--dynamic-width photos-image ipc-sub-grid-item ipc-sub-grid-item--span-2" role="group">
    <div class="ipc-media ipc-media--photo ipc-image-media-ratio--photo ipc-media--base ipc-media--photo-m ipc-photo_photo-image ipc-media_img" style="width:100%">
      
    </div>
  </div>
  <div data-testid="photos_image_overlay_1" class="ipc-lockup-overlay ipc-focus
```

- **Web Scraping – Selenium 4**
- **Baixando imagens**

- Ao inspecionar as imagens, percebemos que elas usam a **tag img**

Imagem de exemplo (Dungeons & Dragons 2: O Poder Maior) com uma sobreposição de informações de inspeção de elementos.

Informações de inspeção de elementos:

- Elemento: `img.ipc-image` (145.5 x 145.5)
- HTML: ``
- URL: `https://m.media-amazon.com/images/M/MV5BMTM3ODc0MDQ5Ni5BMTI5BnBnXkFtZTcwMzcwODQ5Ng@@_V1_QL75_UY280_CR70,0,280,280.jpg`
- Rendered size: 146 x 146 px
- Rendered aspect ratio: 1:1
- Intrinsic size: 280 x 280 px
- Intrinsic aspect ratio: 1:1
- File size: 16.2 kB
- Current source: `https://m.media-amazon.com/images/M/MV5BMTM3ODc0MDQ5Ni5BMTI5BnBnXkFtZTcwMzcwODQ5Ng@@_V1_QL75_UY280_CR70,0,280,280.jpg`

Imagem de exemplo (Carnival Row) com uma sobreposição de informações de inspeção de elementos.

Informações de inspeção de elementos:

- Elemento: `img.ipc-image` (145.5 x 145.5)
- HTML: ``
- URL: `https://m.media-amazon.com/images/M/MV5BMTM3ODc0MDQ5Ni5BMTI5BnBnXkFtZTcwMzcwODQ5Ng@@_V1_QL75_UY280_CR70,0,280,280.jpg`
- Rendered size: 146 x 146 px
- Rendered aspect ratio: 1:1
- Intrinsic size: 280 x 280 px
- Intrinsic aspect ratio: 1:1
- File size: 16.2 kB
- Current source: `https://m.media-amazon.com/images/M/MV5BMTM3ODc0MDQ5Ni5BMTI5BnBnXkFtZTcwMzcwODQ5Ng@@_V1_QL75_UY280_CR70,0,280,280.jpg`

## Web Scraping – Selenium 4

### Baixando imagens

- Primeiramente, vamos importar as bibliotecas necessárias e instanciar o driver.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
import urllib.request

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

driver.get("https://www.imdb.com/title/tt0406728/?ref_=tt_sims_tt_i_1")
driver.implicitly_wait(10) #segundos
```

- Note que como as imagens podem demorar para carregar, vamos utilizar o método `implicitly_wait`

## Web Scraping – Selenium 4

### Baixando imagens

- Posteriormente, capturamos o Xpath e salvamos em uma variável. Por meio desta variável buscamos as imagens específicas com o `find_elements`.

```
div_imagens = driver.find_element(By.XPATH,  
'//*[@id="__next"]/main/div/section[1]/div/section/div/div[1]/section[2]/div[2]/div[2]')  
imagem = div_imagens.find_elements(By.TAG_NAME, "img")[5]  
src = imagem.get_attribute("src")  
print(src)  
  
try:  
    urllib.request.urlretrieve(src, r"C:\Users\Emerson  
Abraham\PycharmProjects\web_scraping\teste.jpg")  
    print("Imagem copiada")  
except:  
    print("Erro")
```

## Web Scraping – Selenium 4

### Trabalhando com inputs em pesquisas.

- Formulários de busca possuem um campo tipo **input**
- O atributo **name** é muito utilizado em campos do tipo **input** que estão em formulários (**form**).
- Com o **find\_element(By.name)** conseguimos buscar o elemento e enviar um dado para o campo.

**Nota:** tenha preferência pelo ID nas buscas, pois este normalmente é um campo único e evita *exceptions*



## Web Scraping – Selenium 4

### Persistência com SQLite



- SQLite é uma biblioteca de linguagem C que implementa um mecanismo de banco de dados SQL pequeno, rápido, independente, de alta confiabilidade e completo.
- É o mecanismo de banco de dados mais usado no mundo, embutido em telefones celulares e na maioria dos computadores.
- SQLite é de domínio público; gratuito para todos usarem para qualquer finalidade.

## Web Scraping – Selenium 4

### Persistência com SQLite

Vamos elaborar uma aplicação simples que faz a persistência de um produto no banco de dados. Crie a classe Product com os atributos id, nome e preço.

```
class Product:

    def __init__(self, id, name,
price):
        self.__id = id
        self.__name = name
        self.__price = price
```

```
def get_id(self):
    return self.__id

def set_name(self, name):
    self.__name = name

def get_name(self):
    return self.__name

def set_price(self, price):
    self.__price = price

def get_price(self):
    return self.__price
```

- **Web Scraping – Selenium 4**
- **Persistência com SQLite**

- Na sequência vamos criar um script para as operações. Primeiramente, fazemos os imports e as funções para abrir a e fechar a conexão com o banco de dados.

```
import sqlite3
from Product import Product

schema = "schema.db"

def open_connection():
    connection = None
    try:
        connection = sqlite3.connect(schema)
    except sqlite3.Error as e:
        print("Erro:", e)

    return connection
```

```
def
close_connection(connection):
    if connection:
        connection.close()
```

- **Web Scraping – Selenium 4**
- **Persistência com SQLite**
- - + • Agora vamos definir a função para criar a tabela e o comando SQL que será enviado para esta função.

```
def create_table(connection, sql_create_table):  
    try:  
        cursor = connection.cursor()  
        cursor.execute(sql_create_table)  
        connection.commit()  
    except sqlite3.Error as e:  
        print("Erro:", e)
```

```
sql_create_table = """CREATE TABLE IF NOT EXISTS products (  
    id integer PRIMARY KEY AUTOINCREMENT,  
    name text NOT NULL,  
    price integer NOT NULL  
);"""
```

- **Web Scraping – Selenium 4**
- **Persistência com SQLite**

• + •

+ •

- Na sequência vamos definir as funções para fazer o CRUD na tabela. Inserir, alterar, excluir e buscar registros (produtos).

|

+ •

- **Inserir**

```
def insert_product(connection, sql_insert_product):  
    try:  
        cursor = connection.cursor()  
        cursor.execute(sql_insert_product)  
        connection.commit()  
    except sqlite3.Error as e:  
        print("Erro:", e)
```



- **Web Scraping – Selenium 4**
- **Persistência com SQLite**
- + •
- + • **Alterar**

```
def update_product(connection, product):
    sql_update_product = "UPDATE products SET name = ?, price = ? WHERE id=?"
    commands=[product.get_name(), product.get_price(), product.get_id()]
    try:
        cursor = connection.cursor()
        cursor.execute(sql_update_product, commands)
        connection.commit()
    except sqlite3.Error as e:
        print("Erro:", e)
```

- **Web Scraping – Selenium 4**
- **Persistência com SQLite**
- + •
- + • • **Excluir**
- 

```
def delete_products(connection):  
    sql_delete_products = "DELETE FROM products"  
    try:  
        cursor = connection.cursor()  
        cursor.execute(sql_delete_products)  
        connection.commit()  
    except sqlite3.Error as e:  
        print("Erro:", e)
```

- Web Scraping – Selenium 4
- Persistência com SQLite

• + •

## + • • Buscar

```
def select_products(connection):
    sql_select_products = "SELECT * FROM products order by price"
    products = None
    try:
        cursor = connection.cursor()
        cursor.execute(sql_select_products)
        products = cursor.fetchall()
    except sqlite3.Error as e:
        print("Erro:", e)
    finally:
        return products
```

- • **Web Scraping – Selenium 4**
- • **Persistência com SQLite**
- +
  - Finalmente, vamos testar; abrir a conexão, criar a tabela e inserir registros no banco
- + •

```
# Abrindo a Conexão  
connection = open_connection()
```

```
create_table(connection, sql_create_table)
```

```
# Inserindo produtos
```

```
sql_insert_product_1 = "INSERT INTO products (name, price) VALUES ('Lego Classic', 126.5)"
```

```
sql_insert_product_2 = "INSERT INTO products (name, price) VALUES ('Cubo Mágico', 50)"
```

```
sql_insert_product_3 = "INSERT INTO products (name, price) VALUES ('Boneco Capitão  
América', 140.35)"
```

```
insert_product(connection, sql_insert_product_1)
```

```
insert_product(connection, sql_insert_product_2)
```

```
insert_product(connection, sql_insert_product_3)
```

## Web Scraping – Selenium 4

### Persistência com SQLite

Por fim, faremos mais algumas simulações e fecharemos a conexão com o banco.

```
# Removendo produtos
delete_products(connection)

# Alterando produtos
product_1 = Product(20, 'Cubo Mágico 3D', 30)
update_product(connection, product_1)

# Buscando produtos
products = select_products(connection)

# Mostrando os produtos
for product in products:
    print(f'Cod: {product[0]} - Produto {product[1]} - preço: R$ {product[2]}')

# Fechando a Conexão
close_connection(connection)
```



# Web Scraping – Selenium 4

## Mini projeto final

Nesse momento, vamos unir todo conhecimento adquirido com este módulo.

1. Faça escolha de um site e inspecione; tenha preferência por aqueles que possuem campos com id e não necessitam validações complicadas.
2. Especifique o que pretende coletar e faça o scraping
3. Salve os dados em um banco de dados.
4. Faça consultas com filtros e exiba os resultados; elabore informações por meio dos dados.

## Web Scraping – Considerações Finais

- Embora seja praticamente “impossível” construir um site a prova de scrapers, muitos deles possuem mecanismos para dificultar esse processo.

Os principais recursos usados são:

- Campos “ocultos” – permitem que o valor contido seja visível apenas no navegador.
- Para “driblar” esses recursos, utilizamos cabeçalhos (HEADERS) bem definidos, bibliotecas e frameworks, tais como BeautifulSoup e Selenium.
- CAPTCHAS e processamento de imagens
- Estes recursos demandam a utilização de ferramentas de reconhecimento de imagem e linguagem natural, tais como bibliotecas de Optical Character Recognition (OCR).

## Web Scraping – Considerações Finais

### Dicas para parecer um ser humano:

- Tempo: certifique-se que seus scrapers não estejam se movendo rápido demais nas páginas (este é um dos principais motivos que levam a identificação de scrapers e colocação em listas negras). Sempre adicione pausas aos scrapers
- Copie os cabeçalhos do seu navegador e adicione a aplicação.
- Certifique-se que não acessará nada que um ser humano não seria capaz de acessar (campos ocultos ao usuário no navegador) – **Honeypots**
- Um acesso a um link “oculto” pode disparar um script do lado do servidor que bloqueará o IP do usuário.

## Web Scraping – Aspectos legais e éticos

- Em 2010, um engenheiro de software coletou dados de aproximadamente, 200 milhões de usuários do Facebook.
- Apesar do web scraping não ser ilegal, devemos nos atentar aos aspectos éticos e legais.
- As principais considerações são:
  - Marca registrada e direitos autorais
  - Invasão de bens móveis
  - Lei geral de proteção de dados (LGPD)

## Web Scraping – Aspectos legais e éticos

- **Marca registrada e direitos autorais**

- ✓ Uma marca registrada é um símbolo, logotipo, palavra ou frase que identifica um produto ou serviço. Uma possível infração a uma marca ocorre quando pessoa ou empresa utiliza sem autorização do proprietário, causando confusão e diluição da marca original.
- ✓ Os direitos autorais protegem obras intelectuais como livros, músicas, filmes e softwares. A lei brasileira 9.610 regula os direitos do autor, sendo que o Art. 5º Para os efeitos desta Lei, considera: obra literária, artística ou científica.



## Web Scraping – Aspectos legais e éticos

- Invasão de bens móveis

Implica em três critérios

1. Invasão: tida quando existe falta de consentimento.
2. Danos aos servidores: scrapers podem derrubar um site ou limitarem a capacidade de atenderem outros usuários.
3. Responsabilidade: se você escreve o código sabe o que ele faz.

## Web Scraping – Aspectos legais e éticos

- Lei geral de proteção de dados (LGPD)
- ✓ Lei brasileira 13.709 que visa proteger os dados pessoais dos cidadãos e estabelecer regras claras para seu tratamento.

Em resumo, não acesse recursos para os quais você não tenha acesso conferido.

# Web Scraping – Considerações Finais

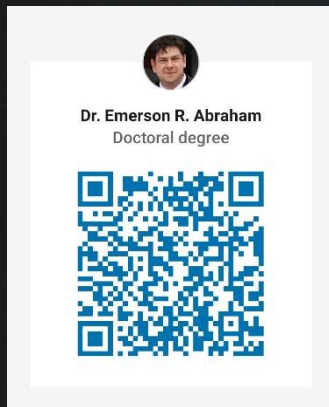
## Roteiro (resumo) para o scraping

- Definir o site do qual se deseja obter informações;
- Requisitar o código HTML da página;
- Inspeccionar e identificar os elementos com as informações a serem coletadas;
- Definir as ferramentas, bibliotecas e frameworks que serão utilizados.
- Efetuar o scraping passo-a-passo, sempre testando.
- Especificar tempo (delay) entre uma operação e outra.
- Salvar os dados no formato desejado.
- Respeitar as leis de direitos autorais e ser consciente quanto a possíveis danos em servidores.

# BIBLIOGRAFIA BÁSICA<sup>+</sup>

- BEAZLEY, David. **Python Essential Reference**, 2009.
- BEK, ANDY. **The Ultimate Web Scraping With Python Bootcamp 2023**.
- BHARGAVA, ADITYA Y. **Entendendo Algoritmos. Um guia ilustrado para programadores e outros curiosos**. São Paulo: Ed. Novatec, 2017
- OCOMOM. Disponível em: <https://ocomonphp.sourceforge.io/>, acessado em 03/2023
- DOCKER. Disponível em : <https://www.docker.com/>, acessado em 03/2023
- DOWNEY, ALLEN B. **Pense em Python. Pense como um cientista da computação**. São Paulo: Ed. Novatec, 2016
- DUMS, Anderson F. **Como logar em uma página web e extrair os dados de uma tabela utilizando python e selenium**, Youtube, 2022
- KOPEC, DAVID. **Problemas clássicos de ciência da computação com Python**. São Paulo: Ed. Novatec, 2019
- MCKINNEY, WILLIAM WESLEY. **Python para análise de dados. Tratamento de dados com Pandas, Numpy e Ipython**. São Paulo: Ed. Novatec, 2018
- MITCHELL, RYAN. **Web Scraping com Python. Coletando mais dados na web moderna**. São Paulo: Ed. Novatec, 2019
- SQLite. Disponível em: <https://www.sqlite.org/index.html>, acessado em: 03/2023
- W3Schools. Disponível em: [https://www.w3schools.com/python/python\\_lists\\_comprehension.asp](https://www.w3schools.com/python/python_lists_comprehension.asp), acesso: 03/2023

# OBRIGADO



## FIAP

Copyright © 2023 | Professor Dr. Emerson R. Abraham

Todos os direitos reservados. A reprodução ou divulgação total ou parcial deste documento é expressamente proibida sem o consentimento formal, por escrito, do professor/autor.



