Universidade do Minho
Departamento de Informática
Mestrado em Engenharia Informática

Mineração de Dados
1º Ano, 2º Semestre
Ano letivo 2025/2026

Ficha de Exercícios PL3
Web Scraping

Duração: 2 horas
Modalidade: Individual ou a pares
Docente: Diana Ferreira

Contexto: Nesta ficha irá aplicar os conceitos fundamentais de Web Scraping com Python.

Setup Inicial:

```
# Criar pasta para a ficha
mkdir md_pl3_scraping
cd md_pl3_scraping

# Criar e ativar ambiente virtual
python -m venv venv
source venv/bin/activate  # Linux/Mac
# venv\Scripts\activate   # Windows

# Instalar dependências
pip install requests beautifulsoup4 pandas selenium lxml jupyter
```

Estrutura de ficheiros recomendada:

```
text
revisao_scraping/
├── exercicio1_http.py
├── exercicio2_seletores.py
├── exercicio3_paginacao.py
├── exercicio4_selenium.py
├── exercicio5_completo.py
├── utils.py # funções auxiliares
└── dados/ # pasta para outputs
```

Universidade do Minho
Departamento de Informática
Mestrado em Engenharia Informática

Parte 1 - Fundamentos HTTP

Site alvo: http://httpbin.org/

Questão 1.1. Complete o seguinte excerto de código python (TODO) de forma a:

- Criar headers HTTP realistas (User-Agent, Accept-Language)
- Executar um pedido GET
- Verificar o status code
- Imprimir os headers enviados (disponíveis na resposta)

```python
import requests
url = "http://httpbin.org/get"
# TODO: Criar headers realistas (User-Agent, Accept-Language)
# TODO: Fazer requisição GET
# TODO: Verificar status code
# TODO: Imprimir os headers enviados (estão na resposta)
```

Questão 1.2. Adapte o código anterior de forma a:

- Extrair o IP do cliente da resposta.
- Verificar se os headers que enviou foram recebidos corretamente.
- Identificar o Content-Type da resposta.

Questão 1.3. Implemente a função abaixo para lidar com falhas de rede:

```python
def requisicao_segura(url, timeout=5, tentativas=3):
    """
    Implementar função que:
    - Tenta a requisição até 'tentativas' vezes
    - Em caso de timeout ou erro, espera e tenta novamente
    - Retorna None se todas falharem
    """
    # TODO: Implementar
    pass
```

Universidade do Minho
Departamento de Informática
Mestrado em Engenharia Informática

Parte 2 - Seletores BeautifulSoup

Site alvo: http://quotes.toscrape.com/

Questão 2.1. Extração com find() e find_all().

Utilizando find_all(), extraia:

- Todas as citações (div.quote)
- Todos os autores (small.author)
- Todas as tags (a.tag)

Utilizando find(), extraia:

- O primeiro autor da página
- A primeira citação
- O link "Next", caso exista

```
import requests
from bs4 import BeautifulSoup

url = "http://quotes.toscrape.com"
response = requests.get(url, headers={'User-Agent': 'Mozilla/5.0'})
soup = BeautifulSoup(response.text, 'lxml')

# TODO: Usando find_all(), extraia:
# - Todas as citações (div.quote)
# - Todos os autores (small.author)
# - Todas as tags (a.tag)

# TODO: Usando find(), extraia:
# - O primeiro autor da página
# - A primeira citação
# - O link "Next" (se existir)
```

Questão 2.2. Extração com select() e select_one().

- Refaça as extrações anteriores utilizando **seletores CSS**
  - select() para múltiplos elementos
  - select_one() para elementos únicos
- Utilize seletores hierárquicos para extrair:
  - Todas as tags **dentro de cada citação**
  - O autor da **primeira citação**, usando hierarquia CSS

Questão 2.3. Extração de atributos. Para cada citação, extraia também:

- O valor do atributo href do link About do autor
- O número total de tags associadas à citação
- O conteúdo do atributo class da citação

Parte 3 - Paginação e Crawling

Site alvo: http://books.toscrape.com/

Questão 3.1. Inspecione a primeira página e identifique os seletores CSS para:

- Livro
- Título
- Preço
- Disponibilidade
- Rating

```
import requests
from bs4 import BeautifulSoup
import time
import random
url_base = "http://books.toscrape.com/catalogue/page-{}.html"
# TODO: Inspecione a primeira página e identifique:
# - Seletor para livros (article.product_pod)
# - Seletor para título (h3 > a)
# - Seletor para preço (p.price_color)
# - Seletor para disponibilidade (p.instock)
# - Seletor para rating (p.star-rating)
```

Universidade do Minho
Departamento de Informática
Mestrado em Engenharia Informática

Questão 3.2. Implemente a função seguinte:

```
def extrair_livros(soup):
    """
    Extrai todos os livros de uma página BeautifulSoup
    Retorna lista de dicionários com:
    - titulo
    - preco
    - disponibilidade (True/False)
    - rating (1-5)
    - url_relativa
    """
    livros = []
    # TODO: Implementar
    return livros
```

Questão 3.3. Implemente o ciclo de crawling completo:

```
todos_livros = []
pagina = 1

while True:
    url = url_base.format(pagina)
    print(f"A processar página {pagina}...")

    # TODO: Requisição com headers
    # TODO: Parsing com BeautifulSoup
    # TODO: Extrair livros da página
    # TODO: Verificar se existe próxima página
    # Dica: Verifique se há livros na página (se não houver, terminou)

    pagina += 1
    # Delay ético
    time.sleep(random.uniform(1, 3))
```

Questão 3.4. Crie um DataFrame com todos os livros e exporte os dados para CSV e JSON.

```
import pandas as pd
import json

# TODO: Criar DataFrame com todos os livros
# TODO: Exportar para CSV (books.csv)
# TODO: Exportar para JSON (books.json)

# TODO: Análise simples:
# - Qual o livro mais caro?
# - Quantos livros estão disponíveis?
# - Distribuição de ratings
```

Parte 4 - Sites Dinâmicos com Selenium

Site alvo: http://quotes.toscrape.com/js/

Questão 4.1. Importe o Selenium e configure Chrome options para headless:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.chrome.options import Options
from bs4 import BeautifulSoup
import time

# TODO: Configurar Chrome options para headless
options = Options()
options.add_argument("--headless")
options.add_argument("--no-sandbox")
options.add_argument("--disable-dev-shm-usage")
```

Questão 4.2. Extração com Selenium puro

```
driver = webdriver.Chrome(options=options)

try:
    driver.get("http://quotes.toscrape.com/js/")

    # TODO: Usar WebDriverWait para aguardar carregamento
    # Aguardar até que as citações estejam presentes

    # TODO: Extrair usando Selenium
    quotes = driver.find_elements(By.CLASS_NAME, "quote")

    for quote in quotes:
        # Extrair texto, autor e tags
        # Dica: quote.find_element(By.CLASS_NAME, "text")
        pass

finally:
    driver.quit()
```

Questão 4.3. Abordagem híbrida (Selenium + BeautifulSoup)

```
driver = webdriver.Chrome(options=options)

try:
    driver.get("http://quotes.toscrape.com/js/")
    # Aguardar carregamento
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CLASS_NAME, "quote"))
    )
    # Passar HTML para BeautifulSoup
    html = driver.page_source
    soup = BeautifulSoup(html, 'lxml')
    # TODO: Extrair usando BeautifulSoup (mais fácil)

finally:
    driver.quit()
```

Questão 4.4. Paginação em sites dinâmicos

```
driver = webdriver.Chrome(options=options)
todas_citacoes = []

try:
    driver.get("http://quotes.toscrape.com/js/")

    while True:
        # Aguardar carregamento
        WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.CLASS_NAME, "quote"))
        )

        # TODO: Extrair citações da página atual

        # Procurar botão "Next"
        try:
            next_button = driver.find_element(By.CSS_SELECTOR, "li.next > a")
            next_button.click()
            time.sleep(2)  # Aguardar carregamento
        except:
            print("Não há mais páginas")
            break

finally:
    driver.quit()
    print(f"Total de citações: {len(todas_citacoes)}")
```