THE FINISH OFF SCHOOL

Data Scraping

Github : link

Enseignant : Massil Achab

Date : 04/06/2023

Introduction

- Data scraping, also known as web scraping, is the process of extracting data from websites.

- Data scraping is used in a variety of fields, including data science, journalism, marketing, and more.

- Use-cases:
    - In real estate: find out the prices of houses in a region
    - In e-commerce: find out at what price your competitors are selling a product

Legal and Ethical Considerations

- Always respect the website's terms of service and consider the ethical implications when scraping data.

- Most websites that care about their data have anti-scraping policies.

- In practice, if they detect that you’re a bot, they will block you or display an error page.

Python

- Python: A powerful, versatile programming language popular in data science.

- Today, we'll play with the following libraries:
  - Requests
  - BeautifulSoup
  - Scrapy
  - Selenium
  - Pandas
  - Smtplib

Python Library – Requests (1/2)

- Requests is a simple, yet elegant HTTP library for Python, built for human beings.

- Requests allows you to send HTTP/1.1 requests with various methods like GET, POST, and others. With it, you can add content like headers, form data, multipart files, and parameters to HTTP requests.

- Feature-rich: Cookies are preserved between requests in a session, sessions with Cookie persistence, browser-style SSL verification, automatic content decoding, basic/digest authentication, elegant key/value cookies, and more.

- Simplicity: The most important feature of requests is that it abstracts the complexities of making requests behind a beautiful, simple API, allowing you to send HTTP requests using a single line of code.

Python Library - Requests (2/2)

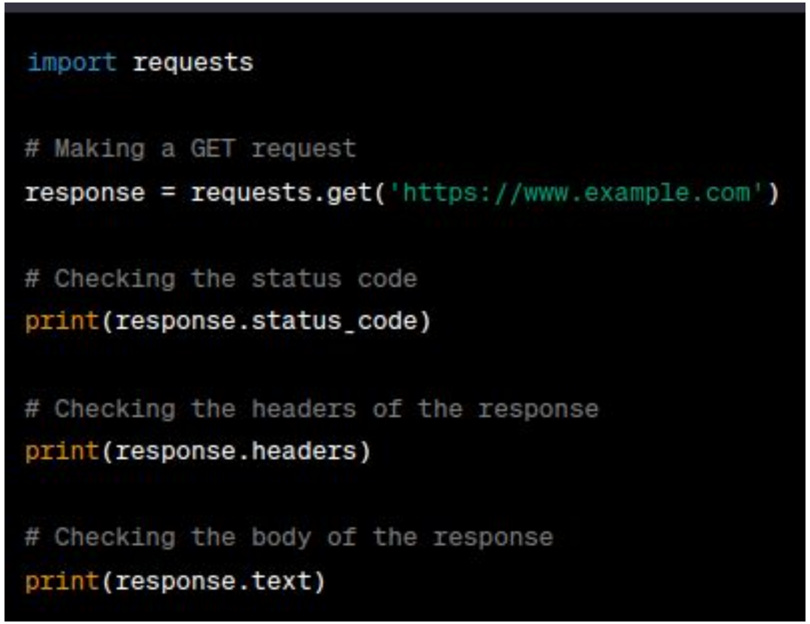
- Import the requests library.

- Make a GET request to a URL using requests.get method. This method returns a Response object.

- Check the status code of the response using the status_code property of the Response object. A status code of 200 means the request was successful.

- Check the headers of the response using the headers property of the Response object. This returns a dictionary of all response headers.

- Check the body of the response using the text property of the Response object. This contains the content of the response.



```python
import requests

# Making a GET request
response = requests.get('https://www.example.com')

# Checking the status code
print(response.status_code)

# Checking the headers of the response
print(response.headers)

# Checking the body of the response
print(response.text)
```

Python Library – BeautifulSoup (1/2)

- BeautifulSoup is a Python library used for parsing HTML and XML documents. It creates parse trees that are helpful to extract the data easily.

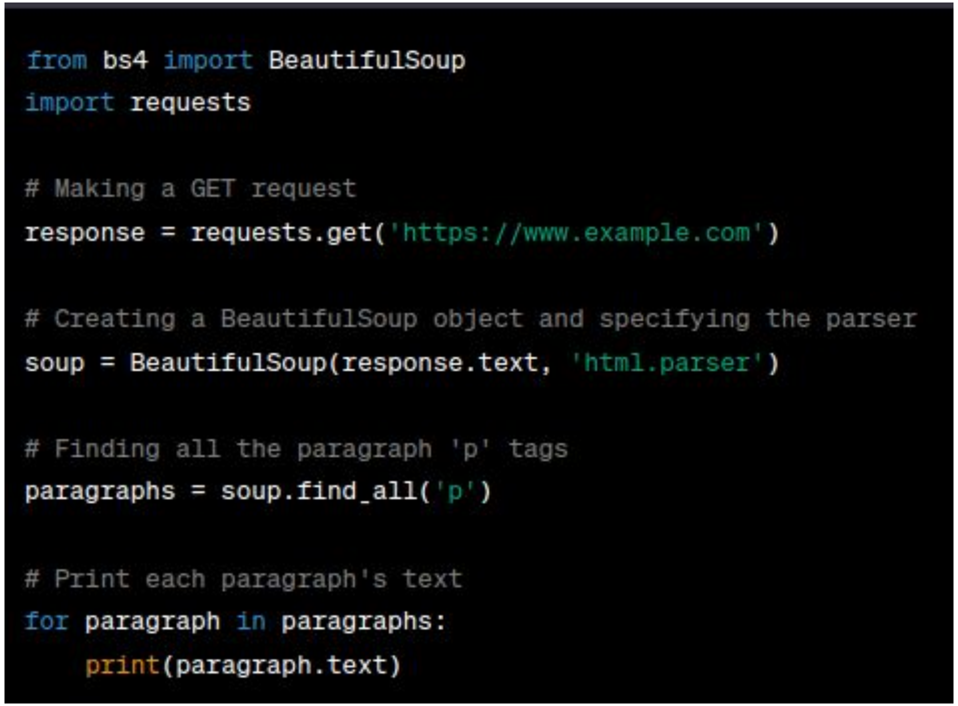
- BeautifulSoup automatically converts incoming documents to Unicode and outgoing documents to UTF-8.

- It allows you to try out different parsers to see which one works best with your specific document. It provides simple, idiomatic ways of navigating, searching, and modifying the parse tree.

- It provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree: a toolkit for extracting what you need.

Python Library – BeautifulSoup (2/2)

- Import the required libraries.
- Make a GET request to a URL using the requests.get method.
- Create a BeautifulSoup object and specify the parser. Here, we're using Python's built-in HTML parser.
- Use the find_all method to find all instances of a certain HTML element. Here, we're finding all paragraph 'p' tags.
- Loop through the found elements and print their text.



```python
from bs4 import BeautifulSoup
import requests

# Making a GET request
response = requests.get('https://www.example.com')

# Creating a BeautifulSoup object and specifying the parser
soup = BeautifulSoup(response.text, 'html.parser')

# Finding all the paragraph 'p' tags
paragraphs = soup.find_all('p')

# Print each paragraph's text
for paragraph in paragraphs:
    print(paragraph.text)
```

Python Library - Scrapy (1/2)

- Scrapy is a powerful, open-source Python library used for web scraping and data extraction from websites.
- Scrapy is built on the Twisted asynchronous networking library, that enables a lot of requests concurrently.
- Scrapy handles large scraping tasks, from simple to complex ones: multiple websites, proxies, and more.
- It respects the rules set in robots.txt and can set download delays to avoid harming websites being scraped.

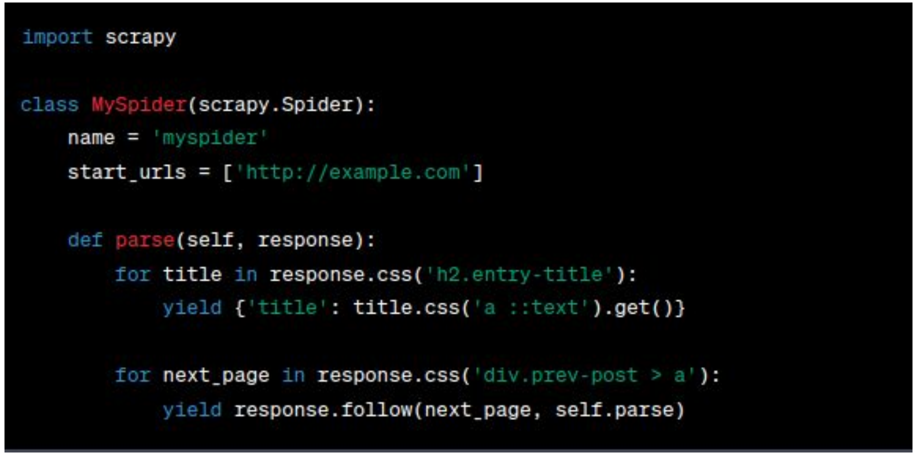
Python Library – Scrapy (2/2)

- A Scrapy Spider is a Python class where you define a custom behavior for crawling and parsing pages.

- Default callback `parse` used by Scrapy to process downloaded responses.

- The yield keyword is used to return the scraped data as a dictionary.

- Scrapy can follow links to other pages by calling the response.follow method.



```
import scrapy

class MySpider(scrapy.Spider):
    name = 'myspider'
    start_urls = ['http://example.com']

    def parse(self, response):
        for title in response.css('h2.entry-title'):
            yield {'title': title.css('a ::text').get()}

        for next_page in response.css('div.prev-post > a'):
            yield response.follow(next_page, self.parse)
```

Python Library – Selenium (1/2)

- Selenium is used for automating web browsers, and for dealing with dynamic websites while scraping.

- Selenium is useful when the data you need to scrape is loaded or generated dynamically with JavaScript.

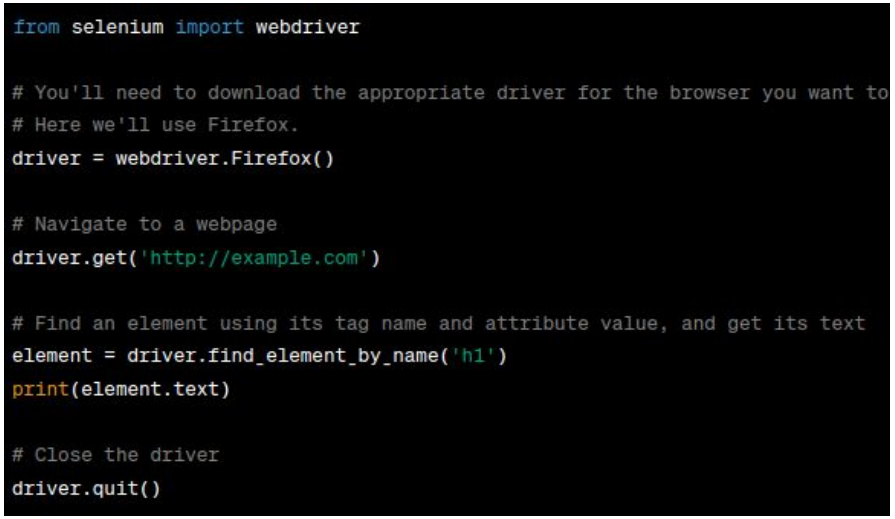
- It can simulate human-like interaction with web pages, like clicking buttons, filling out forms, or scrolling.

- It can be used with headless browsers (browsers without a GUI). It is used for server-side automation tasks.

- Also a tool for testing web apps, due to its ability to automate and replicate user interactions on a webpage.

Python Library - Selenium (2/2)

- Import the webdriver module from selenium.
- Create an instance of the Firefox webdriver to initialize a browser session.
- Use the get method to navigate to a webpage.
- Use the find_element_by_* methods to find HTML elements on the page.
- Once you have an element, you can access its properties, such as its text content.
- Finally, remember to close the driver once you're done using it to free up resources.



```python
from selenium import webdriver

# You'll need to download the appropriate driver for the browser you want to
# Here we'll use Firefox.
driver = webdriver.Firefox()

# Navigate to a webpage
driver.get('http://example.com')

# Find an element using its tag name and attribute value, and get its text
element = driver.find_element_by_name('h1')
print(element.text)

# Close the driver
driver.quit()
```

Working with APIs

- Example of using an API to get data

- Common issues when working with APIs
  - APIs usually allow a limited number of requests per second.
  - If you try to get more, you'll be rate-limited.
  - Relying on an external API adds risk to your project.
  - Your project should be resilient towards API failures, data precision, latency, etc.

Data cleaning

- Why data cleaning is important
    - Some fields may be missing (price of an apartment)
    - Some fields may be incorrect or imprecise (price written in dinar cents instead of dinars)

- Pandas is a Python library for data analysis used for data cleaning and transformation, numerical tables, and time series manipulation.

- Pandas library will be used as well in data engineering courses this week.

Data cleaning with Pandas (1/2)

- Pandas provides two key data structures - "Series" (1-dimensional) and "DataFrame" (2-dimensional), which handle the majority of typical use cases in finance, statistics, social science, and engineering.

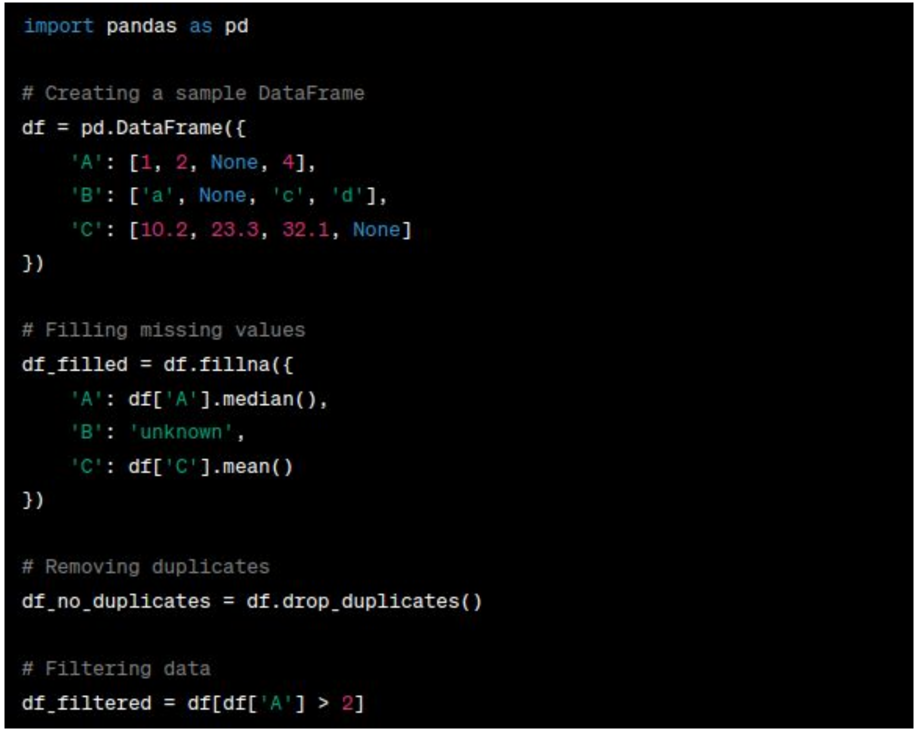
- Pandas provides extensive facilities for data cleaning, filling missing values, removing duplicates, and filtering data, among other functions.

- Pandas is highly optimized for performance, with critical code paths written in Cython or C.

- Pandas can be seamlessly integrated with other libraries in the data science ecosystem of Python like NumPy, Matplotlib, and scikit-learn.

Data cleaning with Pandas (2/2)

- Import the pandas library under the alias pd.
- Create a DataFrame with some missing values.
- Use the fillna method to fill missing values. Here, we're filling missing values in:
  - column 'A' with its median,
  - column 'B' with the string 'unknown',
  - column 'C' with its mean.
- Use the drop_duplicates method to remove duplicated rows.
- Filter rows to only include rows where the value in column 'A' is greater than 2.



```python
import pandas as pd

# Creating a sample DataFrame
df = pd.DataFrame({
    'A': [1, 2, None, 4],
    'B': ['a', None, 'c', 'd'],
    'C': [10.2, 23.3, 32.1, None]
})

# Filling missing values
df_filled = df.fillna({
    'A': df['A'].median(),
    'B': 'unknown',
    'C': df['C'].mean()
})

# Removing duplicates
df_no_duplicates = df.drop_duplicates()

# Filtering data
df_filtered = df[df['A'] > 2]
```

Data storage

- Important to store data somewhere and maintain a history.

- It’s better to store raw data (something even html):
    - to be able to reprocess it later
    - to extract new fields from html pages later

- Different ways to store scraped data: CSV, SQL databases, NoSQL databases

Data storage – CSV files

- Think about a consistent structure to store data

- Writing scraped data to CSV files using Python's built-in functions or pandas.DataFrame.to_csv()

- Advantages and disadvantages of CSV storage
    - Pro: Easy to understand, can be read by Excel
    - Con: No compression
    - Con: No typed fields (a number can be read as a string, a date can be read as a string, etc.)

- Better to use Parquet that address both problems above.

Data storage – Databases

- Storing scraped data in databases

- Introduction to storing data in SQL or NoSQL databases
  - SQL is better for performance, simpler for processing data.
  - NoSQL is more flexible.

- Basic example of using Python libraries (like psycopg or sqlalchemy) to store data in a database

- Considerations when choosing database storage

Data storage - Real-time notification (1/2)

- smtplib is a Python library for sending emails using the Simple Mail Transfer Protocol (SMTP). It is included in the Python Standard Library, it does not require any additional installation.

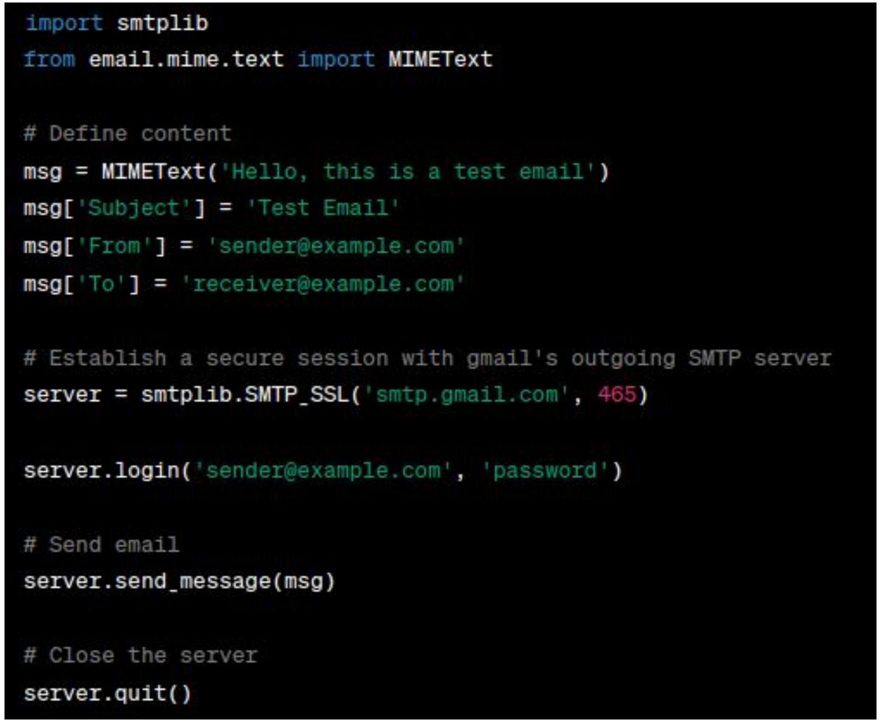
- smtplib defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or Extended SMTP listener daemon.

- The smtplib module can be used to send emails with plain text or HTML content, and also supports file attachments.

- It can send emails via SMTP servers that need authentication, and can encrypt connections with SSL/TLS.

Data storage - Real-time notification (2/2)

- Import the smtplib library and MIMEText from email.mime.text.
- Create an email message using MIMEText. This includes setting the email content and headers.
- Establish a secure connection with the outgoing SMTP server using smtplib.SMTP_SSL.
- Log into the SMTP server using your email address and password.
- Send the email using the send_message method of the SMTP server object.
- Close the connection to the SMTP server using the quit method.



```python
import smtplib
from email.mime.text import MIMEText

# Define content
msg = MIMEText('Hello, this is a test email')
msg['Subject'] = 'Test Email'
msg['From'] = 'sender@example.com'
msg['To'] = 'receiver@example.com'

# Establish a secure session with gmail's outgoing SMTP server
server = smtplib.SMTP_SSL('smtp.gmail.com', 465)

server.login('sender@example.com', 'password')

# Send email
server.send_message(msg)

# Close the server
server.quit()
```

Web scraping challenges

- Dealing with dynamic content: AJAX, JavaScript rendering
- Handling CAPTCHAs and login pages
- Dealing with rate limiting and IP blocking
- Deduplication of similar products

Web scraping best practices

- Respectful scraping: following robots.txt, setting reasonable request rates

- Tips for efficient scraping:
  - caching
  - parallelization
  - different IP addresses using proxies
  - error handling to avoid breaking

Real-world example

- Full example of a web scraping project from start to finish
- Project idea, planning, and execution
- Dealing with challenges, cleaning and storing the data

Questions?