📦 Using `requests` Module for Data Collection

Web scraping is a powerful technique to collect data from websites. In this guide, we'll explore how to use Python's `requests` module to fetch the raw HTML content of web pages — a foundational step in any web scraping workflow.

We will be using two safe demo sites for practice:

- https://quotes.toscrape.com/
- https://books.toscrape.com/

1️⃣ What is `requests`?

The `requests` module is a **Python library** used to send **HTTP/HTTPS requests**. It provides a simple API for interacting with web services or downloading web pages.

- Helps you fetch content of a webpage.
- Commonly used before parsing HTML with tools like `BeautifulSoup`.

2️⃣ Installing `requests`

To install the `requests` library, run the following command in your terminal or command prompt:

bash
CopyEdit
```
pip install requests
```

3️⃣ Sending a Basic GET Request

To fetch a webpage, send a GET request like this:

python
CopyEdit
```
import requests

url = "https://example.com"
response = requests.get(url)

# Print the HTML content
print(response.text)
```

🔑 Key Concepts:

- `url`: The website you want to access.
- `response.text`: The webpage's HTML content in string format.

4️⃣ Checking the Response Status

Always verify if your request was successful:

python
CopyEdit
```
print(response.status_code)
```

✅ Common HTTP Status Codes:

Code	Meaning
200	OK (Success)
404	Not Found
403	Forbidden

500	Internal Server Error

✔️ Good Practice:

python
CopyEdit
```python
if response.status_code == 200:
    print("Page fetched successfully!")
else:
    print("Failed to fetch the page.")
```

5️⃣ Important Response Properties

Property	Description
`response.text`	HTML content as Unicode text
`response.content`	Raw bytes of the response
`response.status_code`	HTTP status code
`response.headers`	Metadata like content-type, server info

6️⃣ Adding Headers to Mimic a Browser

Some websites block bots. To avoid detection, use headers like:

python
CopyEdit
```python
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
}

response = requests.get(url, headers=headers)
```

7️⃣ Handling Connection Errors

Use a `try-except` block to prevent crashes:

python
CopyEdit
```python
try:
    response = requests.get(url, timeout=5)
    response.raise_for_status()  # Raises error for bad responses
    print(response.text)
except requests.exceptions.RequestException as e:
    print(f"An error occurred: {e}")
```

8️⃣ Best Practices for Fetching Pages

✔️ **Check status codes** to ensure success
✔️ **Use headers** to appear like a browser
✔️ **Set a timeout** to avoid hanging requests
✔️ **Respect website limits** (avoid sending too many requests too fast)

9️⃣ Summary

- The `requests` module simplifies HTTP requests in Python.
- It's the **first step in most scraping projects**.
- Often combined with libraries like **BeautifulSoup** or **lxml** for parsing and extracting data from HTML.