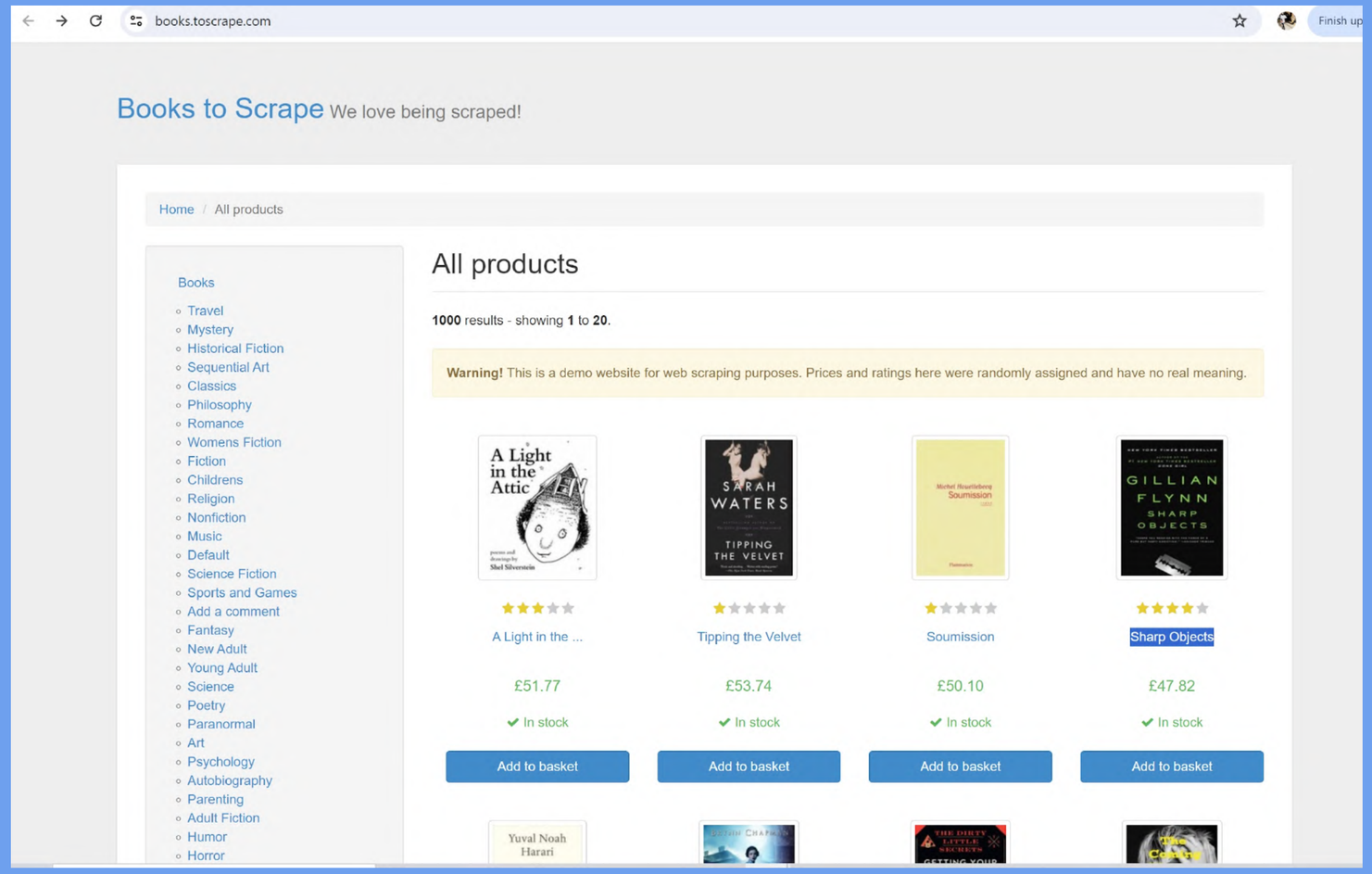
Web Scraping

Hebah Arafeh

Introduction to Web Scraping

What is web scraping?
Why scrape book data?



INTRODUCTION TO WEB SCRAPING

What is Web Scraping?

- Web scraping is the process of automatically extracting information from websites.
- It involves fetching the web pages and parsing the HTML to gather the desired data.

Why Scrape Book Data?

- To collect information on books such as titles, prices, availability, and ratings.
- Useful for creating a searchable database of books.
- Can be applied for market research, price comparison, or personal collection management.

Tools and Libraries

Python Libraries Used:
requests:
Makes HTTP requests to fetch web pages.
Simple API for sending GET and POST requests.
BeautifulSoup (bs4):
Parses HTML and XML documents.
Extracts data from web pages with ease.
sqlite3:
Built-in Python library for SQLite databases.
Stores and manages the scraped data efficiently.
tkinter:
Standard Python interface to the Tk GUI toolkit.
Creates graphical user interfaces for our application.

Try Pitch

Tools and Libraries

Why These Tools?
requests: Reliable and easy to use for web scraping.
BeautifulSoup: Powerful for parsing HTML and extracting information.
sqlite3: Lightweight and perfect for handling structured data.
tkinter: Simplifies the creation of desktop GUIs.

Step 1: Fetching Web Pages

Using the **requests** library to fetch HTML content

Handling HTTP requests

python

```python
def fetch_page(url):
    response = requests.get(url)
    if response.status_code == 200:
        return response.content
    else:
        print(f"Failed to fetch page: {url}")
        return None
```

```python
# Step 3: Data Parsing with BeautifulSoup
def extract_books_info(html_content):
    soup = BeautifulSoup(html_content, 'html.parser')
    books = []
    for book in soup.find_all('article', class_='product_pod'):
        title = book.h3.a['title']
        price = book.find('p', class_='price_color').get_text().strip()
        availability = book.find('p', class_='instock availability').get_text().strip()
        rating = book.find('p', class_='star-rating')['class'][1]
        books.append({'title': title, 'price': price, 'availability': availability, 'rating': rating})
    return books
```

Step 2: Parsing HTML Content

Using BeautifulSoup to parse HTML
Extracting book details: title, price, availability, and rating

Step 3: Storing Data in SQLite

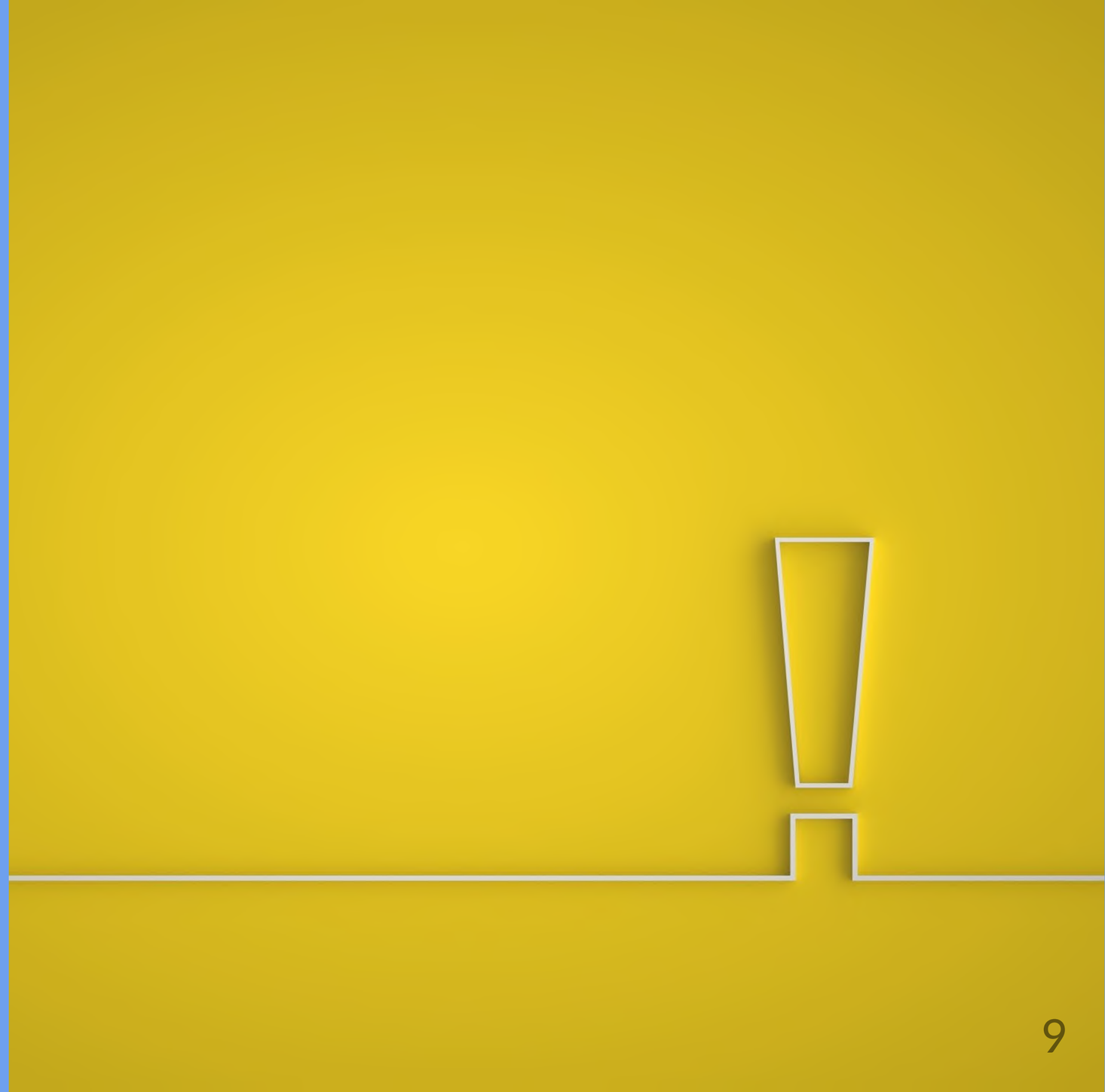
Creating and connecting to the SQLite database
Inserting book data while handling duplicates

```python
# Step 4: Data Storage Using SQLite
def create_database():
    conn = sqlite3.connect('books.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS books
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                 title TEXT UNIQUE,
                 price TEXT,
                 availability TEXT,
                 rating TEXT)''')
    conn.commit()
    conn.close()
```

```python
def insert_books(books):
    conn = sqlite3.connect('books.db')
    c = conn.cursor()
    for book in books:
        try:
            c.execute("INSERT INTO books (title, price, availability, rating) VALUES (?, ?, ?, ?)",
                      (book['title'], book['price'], book['availability'], book['rating']))
        except sqlite3.IntegrityError:
            print(f"Book '{book['title']}' already exists in the database. Skipping.")
    conn.commit()
    conn.close()
```

Step 4: Error Handling

Managing HTTP Errors:
Timeouts: Use retries and timeouts to handle slow or unresponsive servers.
Connection Errors: Catch exceptions to manage network-related errors.
Status Codes: Check and handle different HTTP status codes (e.g., 404, 500).
Handling Duplicates in the Database:
Unique Constraints: Ensure the database schema uses unique constraints to prevent duplicate entries.
Exception Handling: Use try-except blocks to handle SQLite integrity errors when inserting data.



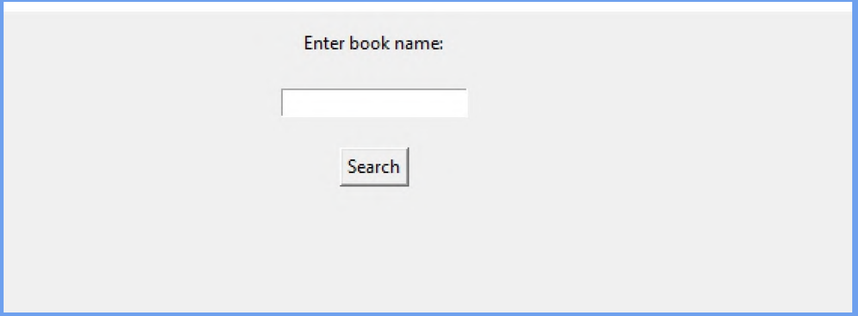
Handling HTTP Errors:

```python
import requests
from requests.exceptions import RequestException, Timeout

def fetch_page(url):
    try:
        response = requests.get(url, timeout=10)
        response.raise_for_status()
        return response.content
    except Timeout:
        print("Request timed out. Retrying...")
        return None
    except RequestException as e:
        print(f"Error fetching page: {e}")
        return None
```

```python
def insert_books(books):
    conn = sqlite3.connect('books.db')
    c = conn.cursor()
    for book in books:
        try:
            c.execute("INSERT INTO books (title, price, availability, rating) VALUES (?, ?, ?, ?)",
                      (book['title'], book['price'], book['availability'], book['rating']))
        except sqlite3.IntegrityError:
            print(f"Book '{book['title']}' already exists in the database. Skipping.")
    conn.commit()
    conn.close()
```

Step 5: Implementing the GUI



Using Tkinter to create a simple user interface
Allowing users to search for books by name

```
def search_books():
    book_name = entry.get()
    results = query_books_by_name(book_name)
    if results:
        result_text = "\n".join([f"Title: {book[1]}, Price: {book[2]}, Availability
        messagebox.showinfo("Search Results", result_text)
    else:
        messagebox.showinfo("Search Results", "No books found matching that name.")
```

Demo Time

Thank You

Pitch

Want to make a presentation like this one?

Start with a fully customizable template, create a beautiful deck in minutes, then easily share it with anyone.

Create a presentation (It's free)