Python Refresher

1. Data Types & Variables

Defined basic Python variables (str, int, float, bool) to represent a book's metadata and printed them to confirm correct types and formatting. This section ensures understadning how core data types work and how to embed them in f-strings.

```
# Python Library Management System


!pip install requests
import pickle, sqlite3, requests
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from reques
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from
```

```
# 1. Data Types and Variables
book_title = "Python Programming"
book_quantity = 5
book_price = 29.99
is_available = True

print(f"Book: {book_title}, Quantity: {book_quantity}, Price: ${book_price}, Available: {is_available}")
```

```
Book: Python Programming, Quantity: 5, Price: $29.99, Available: True
```

2. Lists

A list of book titles was created and then used append() and insert() to add new items at the end and at a specific index. This demonstrates dynamic collection manipulation and how to grow or rearrange lists.

```
# 2. Lists
books = ["Python Basics", "Advanced Python", "Data Science with Python"]

# TODO: Add two more books to the list
books.append("Machine Learning 101")
books.insert(1, "Fluent Python")  # insert at index=1

print("Updated book list:", books)
```

```
t: ['Python Basics', 'Fluent Python', 'Advanced Python', 'Data Science with Python', 'Machine Learni
```

3. Loops & range()

By iterating over range(len(books)), printed each book alongside its index, showing how to combine loop counters with list indexing. This reinforces the pattern of using range() for index-based traversal.

```
# 3. Loops and range()
print("\nAvailable books:")
for i in range(len(books)):
    print(f"{i}: {books[i]}")
```

```

Available books:
0: Python Basics
1: Fluent Python
2: Advanced Python
3: Data Science with Python
4: Machine Learning 101
```

4. User Input

The user is prompted to enter a new book title, then adds that input to the books list and prints the updated list.

```
# 4. User Input
new_book = input("Enter a new book title: ")
books.append(new_book)
print("Updated book list:", books)
```

```
Enter a new book title: AI and Machine Learning for Coders
Updated book list: ['Python Basics', 'Fluent Python', 'Advanced Python', 'Data Science with Python'
```

5. Functions & Conditional Statements

Implemented check_availability() to return True if a book exists in our list and False otherwise, illustrating if…else logic inside a function. This section highlights encapsulation of logic and boolean membership checks.

```
# 5. Functions and Conditional Statements
def check_availability(book_name):
    if book_name in books:
        return True
    else:
        return False
    pass

# Test the function
print(f"Is 'Python Basics' available? {check_availability('Python Basics')}")
```

```
Is 'Python Basics' available? True
```

6. File I/O

The save_books_to_file() and read_books_from_file() functions write the in-memory list to a text file and read it back, handling missing files. Learned how to open files in different modes and process file lines.

```
# 6. File I/O
def save_books_to_file(filename):
    with open(filename, 'w') as f:
        for book in books:
            f.write(f"{book}\n")

def read_books_from_file(filename):
    try:
        with open(filename, 'r') as f:
            return [line.strip() for line in f.readlines()]
    except FileNotFoundError:
        print(f"Error: {filename} not found.")
        return []


# Test file I/O
save_books_to_file("books.txt")
print("Loaded from file:", read_books_from_file("books.txt"))
```

```
'Advanced Python', 'Data Science with Python', 'Machine Learning 101', 'AI and Machine Learning for
```

7. Pickle Serialization

Serialized the list of books to a binary .pkl file and then loaded it back with error handling for missing or corrupted files. This shows how to persist complex Python objects between sessions.

```
# 7. Pickle for serialization
def save_books_pickle(filename):
    with open(filename, 'wb') as f:
        pickle.dump(books, f)

def load_books_pickle(filename):
    try:
        with open(filename, 'rb') as f:
            return pickle.load(f)
    except (FileNotFoundError, pickle.UnpicklingError) as e:
        print(f"Error loading pickle: {e}")
        return []

save_books_pickle("books.pkl")
print("Books loaded from pickle:", load_books_pickle("books.pkl"))
```

```
Books loaded from pickle: ['Python Basics', 'Fluent Python', 'Advanced Python', 'Data Science with
```

8. SQLite Database Interaction

Using sqlite3, created a books table, inserted records (add_book_to_db()), and implemented update_book_in_db() to change title or quantity. This section demonstrates basic relational database operations from Python.

```
# 8. SQLite Database Interaction
def create_books_table():
    conn = sqlite3.connect('library.db')
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS books
                     (id INTEGER PRIMARY KEY, title TEXT, quantity INTEGER)''')
    conn.commit()
    conn.close()

def add_book_to_db(title, quantity):
    try:
        conn = sqlite3.connect('library.db')
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO books (title, quantity) VALUES (?, ?)",
            (title, quantity)
        )
        conn.commit()
    except sqlite3.Error as e:
        print(f"Database error: {e}")
    finally:
        conn.close()

create_books_table()
add_book_to_db("Python for Beginners", 10)
# Verify:
conn = sqlite3.connect('library.db')
print(conn.execute("SELECT * FROM books").fetchall())
conn.close()
```

```
[(1, 'Python for Beginners', 10)]
```

9. Web API Interaction

Queried the GitHub Search API for top Python repositories and parsed the JSON response to extract the top 5 repo names. Demonstrating how to make HTTP requests, handle JSON, and integrate external data sources.

```
# 9. Web API Interaction (GitHub API)
def get_python_repos():
    url = "https://api.github.com/search/repositories"
    params = {"q": "language:python", "sort": "stars", "order": "desc"}
    response = requests.get(url, params=params)
    data = response.json()
    return [repo['name'] for repo in data['items'][:5]]

# Test API function
print("Top 5 Python repositories on GitHub:", get_python_repos())

print("\nLibrary Management System operations completed.")
```

```
Top 5 Python repositories on GitHub: ['free-programming-books', 'public-apis', 'system-design-prime

Library Management System operations completed.
```

10. Remove Book Function

The remove_book() function safely removes a title from the in-memory list, catching ValueError if the book isn't present. This illustrates list modification and exception handling.

```
# 10. Remove a Book Function
def remove_book(book_name):
    """Remove a book from the in-memory list if it exists."""
    try:
        books.remove(book_name)
        print(f"Removed '{book_name}' from library.")
    except ValueError:
        print(f"Error: '{book_name}' not found in library.")

remove_book("Advanced Python")
print("Current books:", books)
```

```
Removed 'Advanced Python' from library.
Current books: ['Python Basics', 'Fluent Python', 'Data Science with Python', 'Machine Learning 101
```

11. Update Book Info in DB

With update_book_in_db(), you can change a book's title and/or quantity in the SQLite database by building a dynamic UPDATE statement. It reinforces parameterized queries to prevent SQL injection and shows conditional SQL generation.

```
# 11. Update Book Infor in the Database
def update_book_in_db(book_id, new_title=None, new_quantity=None):
    """
    Update title and/or quantity of a book in the SQLite DB.
    Provide book_id and at least one of new_title or new_quantity.
    """
    conn = sqlite3.connect('library.db')
    cursor = conn.cursor()
    updates, params = [], []
    if new_title is not None:
        updates.append("title = ?")
        params.append(new_title)
    if new_quantity is not None:
        updates.append("quantity = ?")
        params.append(new_quantity)
    params.append(book_id)
    sql = f"UPDATE books SET {', '.join(updates)} WHERE id = ?"
    cursor.execute(sql, params)
    conn.commit()
    conn.close()
    print(f"Book {book_id} updated.")

update_book_in_db(1, new_quantity=15)
```

```
Book 1 updated.
```

12. CLI Loop Using Input & Menus

The run_library_cli() function offers a simple text menu for viewing, adding, removing, and updating books, then cleanly exiting. This brings together all previous functions into an interactive workflow and demonstrates handling user input and control flow.

```
# 12. Simple CLI Loop with Input & Menus
def run_library_cli():
    """A minimal command-line interface for our library system."""
    create_books_table()  # ensure table exists
    while True:
        print("\nLibrary Menu:")
        print("1. View books")
        print("2. Add book")
        print("3. Remove book")
        print("4. Update book in DB")
        print("5. Exit")
        choice = input("Enter choice [1-5]: ")
        if choice == '1':
            print("In-memory books:", books)
        elif choice == '2':
            t = input("Title: ")
            q = int(input("Quantity: "))
            books.append(t)
            add_book_to_db(t, q)
        elif choice == '3':
            t = input("Title to remove: ")
            remove_book(t)
        elif choice == '4':
            bid = int(input("Book ID to update: "))
            new_q = input("New quantity (leave blank to skip): ")
            new_t = input("New title (leave blank to skip): ")
            update_book_in_db(bid,
                              new_title=new_t or None,
                              new_quantity=int(new_q) if new_q else None)
        elif choice == '5':
            print("Goodbye!")
            break
        else:
            print("Invalid choice. Try again.")

run_library_cli()
```

```

Library Menu:
1. View books
2. Add book
3. Remove book
4. Update book in DB
5. Exit
Enter choice [1-5]: 1
In-memory books: ['Python Basics', 'Fluent Python', 'Data Science with Python', 'Machine Learning 1

Library Menu:
1. View books
2. Add book
3. Remove book
4. Update book in DB
5. Exit
Enter choice [1-5]: 5
```

```
Goodbye!
```