\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CSC221 ADVANCED Python Programming**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

LAB12 DJANGO REST FRAMEWORK – book application

Objectives

In this lab assignment, students will learn:

* Explore the code used to build a Django REST API with Python
* Create a Django REST framework application\*

**PRE-REQs:** Install Django and Django REST framework packages:

* django 🡨 *should already be installed in books\_env*
* djangorestframework
* httpie ~ a command line HTTP client

**Plus** … **sqlite3** ~ an SQLite command line tool [for more info see ADDENDUM]

# Instructions

You will not need to create a new folder folder for this week’s lab. This week’s lab will build a REST API, using the Django REST framework, for the **book** application built in the previous lab.

**NOTE**: You were required to submit a ZIP file of your **django\_books** directory for Lab10. Be sure you have a copy of this file somewhere SAFE, just in case you make a mistake working through this lab exercise. Worst case scenario, you can restore your code to the original starting point using this your **django\_books.zip** file.

(\*) This lab is similar to the Django RESTful API example (Chapter 5) provided by ***Hands-On RESTful Python Web Services – Second Edition*** *by Gaston C. Hillar* (Packt Publishing Ltd.)

A PDF version of the *First Edition* of this book is available for download from **NC LIVE**:

* <https://www.nclive.org/>

The *Second Edition* of this book is available via **GitHub**:

* <https://github.com/PacktPublishing/Hands-On-RESTful-Python-Web-Services-Second-Edition>]
* **Note**: I also found a few links to PDF versions but not all worked.

This lab will cover the modification of the Django app named **book** within the **django\_books** project created in Lab11 to add a REST API.

## **More on the Tutorial …**

You will be expected to complete the application following the instructions provided with ***multiple (8) screen captures*** required for your Lab 12 submission. These screen captures should be “pasted” in the Word document provided

* *Lab12-BookAppScreenCaptures.docx* ~ the document contains placeholders for each of the required screen captures.

As was the case in the previous lab, you will **not** be using Jupyter Notebook to complete this lab. Much of the lab will be done entering commands from the command line via a terminal window and by creating or updating files *using your favorite code editor or IDE*.

### Code Assistance

You will be provided with a few files to assist with this lab:

* **book-api-code.txt** ~ code for the **book** app Python files so you can *copied-and-pasted* directly into your code editor.
* **Python-shell-cmds.txt ~** Python (interactive) shell commands
* **HTTPie-commands.txt ~** HTTPie commands

Please take some time to READ the comments in the code and any additional notes in the lab so you have a better understanding of the code and how to use it. As stated previously, Django’s is the most popular Web and REST framework with larger sites (companies) and is frequently mentioned in job postings, so it is worth learning if you are interested in Python web development and REST APIs.

### **Getting Started**

* Verify you have a SAFE copy of **django\_books.zip** from **Lab11**
* Open a command prompt window in the same folder used for **Lab11**
* Navigate to the location of your **project-books** directory
* Activate your virtual environment (**books\_env**)
  + ~ use the same commands used in Lab11

**Windows Users:**

* books\_env\Scripts\activate.bat | deactivate.bat

**Mac Users:**

* source books\_env/bin/activate | deactivate

### **Updating your Virtual Environment for the Django project**

Once your virtual environment has been activated, install 2 additional packages:

* pip install djangorestframework
* pip install httpie

Command Window showing the results of 2 commands:
- pip install djangorestframework
- pip install httpie

### **LAB 11 REVIEW…**

In the previous lab your created a:

* Django project: **django\_books**
* Django app: **books**

Your directory structure should look like this:

* project-books\django\_books\**books <DIR>**
* project-books\django\_books\**django\_books <DIR>**
* project-books\django\_books\**manage.py**
* project-books\django\_books\**db.sqlite3**

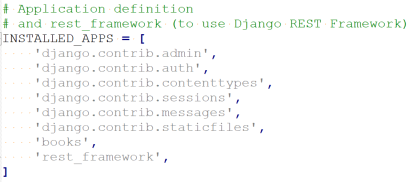
You added the **books** app to the list of INSTALLED\_APPS in:

* project\_books\django\_books\**django\_books\settings.py**

However, we now need to add the REST framework as one of the INSTALLED\_APPS.

**Edit** the project\_books\django\_books\**django\_books\settings.py** file.

* **Add** ‘rest\_framework’



### **Reviewing the Books model definition**

You created the **model** for the data associated with each **Book**.

**View** the project\_books\django\_books\**books\models.py** file

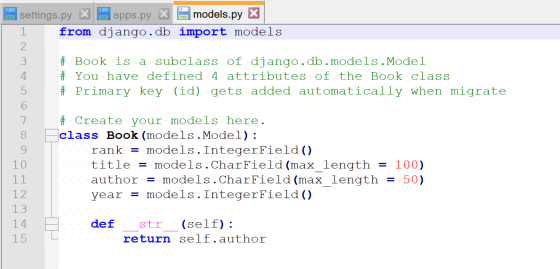


Figure 1 Extra comments added for reference

Review the contents & comments in this file. The models file represents the data persisted for each Book. Each attribute is declared within the Books class, complete with field types, max lengths, defaults, etc. and represents one column in the database.

**Note**: When the database was created, Django automatically added an auto-incremented, integer, primary key column named **id**.

### **Verification of the database table(s)**

**View** the project\_books\django\_books\**django\_books\settings.py** file again.

Look for the DATABASES definition – it already had the ENGINE & NAME correctly defined because we are using the default **db.sqlite3** database.



See **ADDENDUM** section at end of Lab if you have problems running sqlite3

Use **sqlite3** command to view all the tables & books schema created:

* **Enter** the following commands from the command line:
  + sqlite3 db.sqlite3 “.tables”
  + sqlite3 db.sqlite3 “.schema books\_book”

Sample Output:

Command window showing the results of 2 commands: 
- sqlite3 db.sqlite3 “.tables”
- sqlite3 db.sqlite3 “.schema books_book”


**SCREEN CAPTURE # 1 - Command window showing “.tables” & “.schema …” output**

Use **sqlite3** command to view the contents of table **books\_book**:

* **Enter** the following command from the command line:
  + sqlite3 db.sqlite3 “SELECT \* FROM books\_book ORDER BY rank;”

Sample Output:

Command window showing the results of command:
- sqlite3 db.sqlite3 "SELECT * FROM books_book ORDER BY rank;"

**SCREEN CAPTURE # 2 - Command window showing “SELECT \* FROM …” output**

### **Managing serialization & deserialization**

Our RESTful web API must be able to serialize Book instances into JSON … and … it must be able to deserialize JSON to build Book instances. Django uses a two-phase process to do this:

Model (**Book**) 🡪 Serializers 🡪 Python dictionary {data}

Python dictionary {data} 🡪 (JSON)Renderers 🡪 b’{data}

b’{data} 🡪 (JSON)Parsers 🡪 Python dictionary {data}

Deserializers 🡪 Model (**Book**)

#### Serializers

“Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types.” Serialization takes a data structure (i.e model) and converts it into a series of bytes that can be stored or transferred (i.e sent across the wire).

[Ref: <https://www.django-rest-framework.org/api-guide/serializers/>]

#### Deserializers

Deserializers reverse the serialization process, they take a series of bytes and uses them to a recreate the data structure (i.e. model) for use by a program.

#### Renderers

“REST framework includes a number of built in Renderer classes, that allow you to return responses with various media types.” Django provides a variety of renderers, including a JSONRenderer. In addition, there are third-party packages available that work with Django and provide YAML, XML, XLSX, CSV and Pandas renderers.

[Ref: <https://www.django-rest-framework.org/api-guide/renderers/> ]

#### Parsers

“REST framework includes a number of built in Parser classes, that allow you to accept requests with various media types.” As with Renderers, there are third-party packages available that work with Django and provide YAML and XML parsers.

[Ref: <https://www.django-rest-framework.org/api-guide/parsers/> ]

Create the **Serializer** code for the **Book** model.

* **Create** a new project\_books\django\_books\**books\serializers.py** file
* **Copy** the Serializer code provided in **book-api-code.txt** into the file

Review the contents & comments in this file. The serializers file declares the attributes you want to be serialized.

The **create** method receives validated\_data, meaning the data must match the defined attribute requirements (length, type, etc.).

The **update** method receives and existing instance of Book and the new validated\_data for the defined attributes.

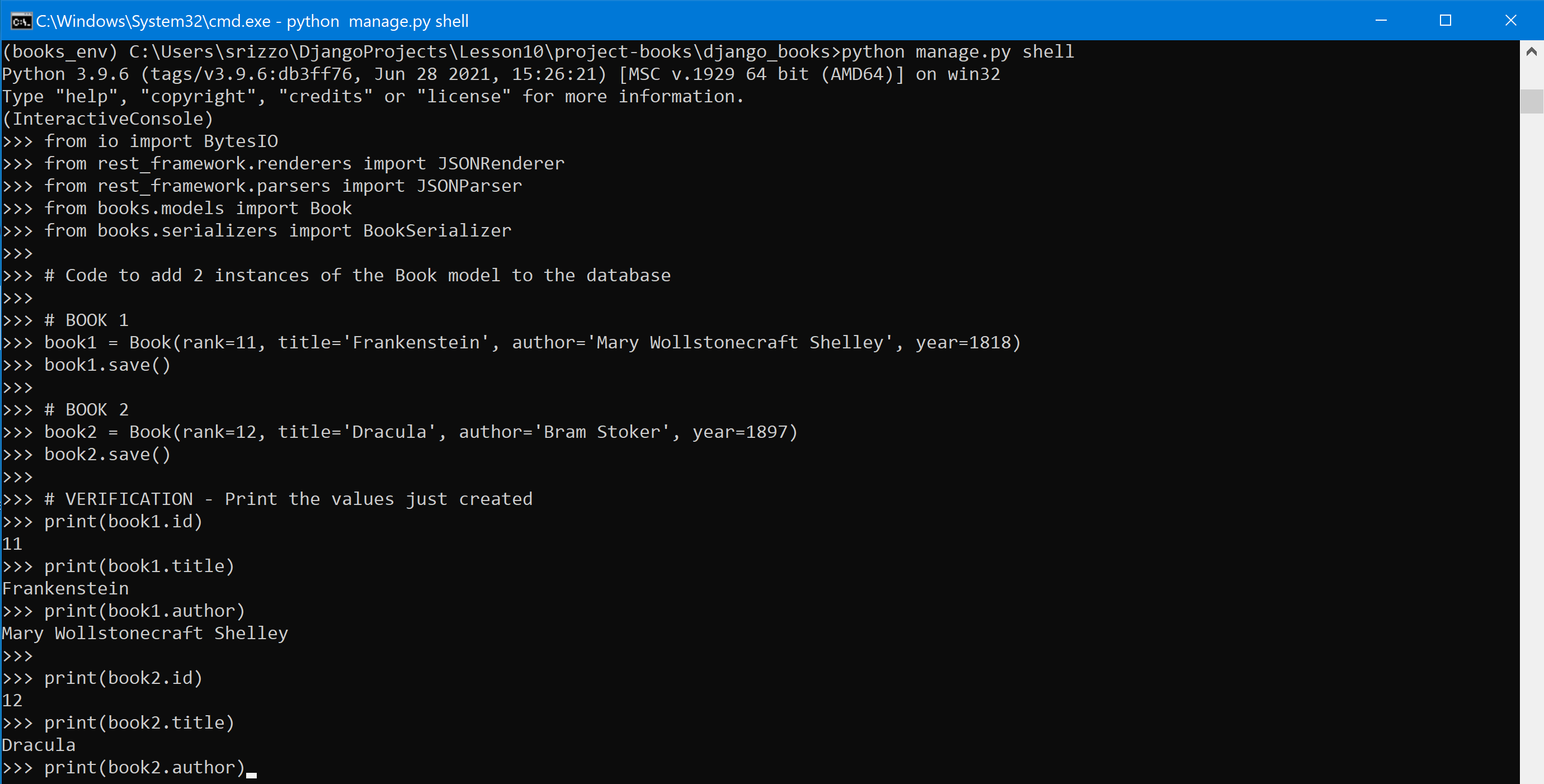
### **testing serialization & deserialization**

Now that our Model and Serializer are complete, we can use the Python interactive shell to run some code to test our Book model and the corresponding Serializer code.

Launch the Python interactive shell:

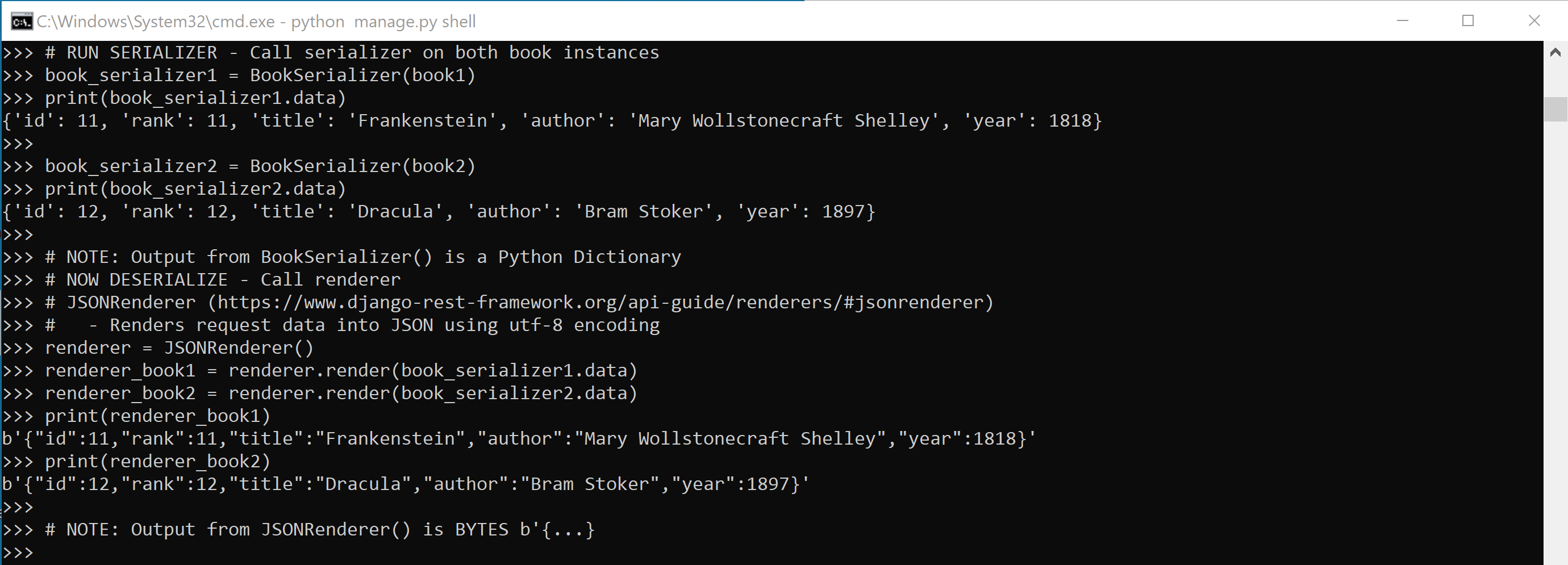
* **Enter** the following command to start the shell:
  + **python manage.py shell**
* **Copy** the code provided in **Python-shell.txt** 
  + copy & run the code “chunk-by-chunk” (recommendation)

Sample Output from CHUNK1:



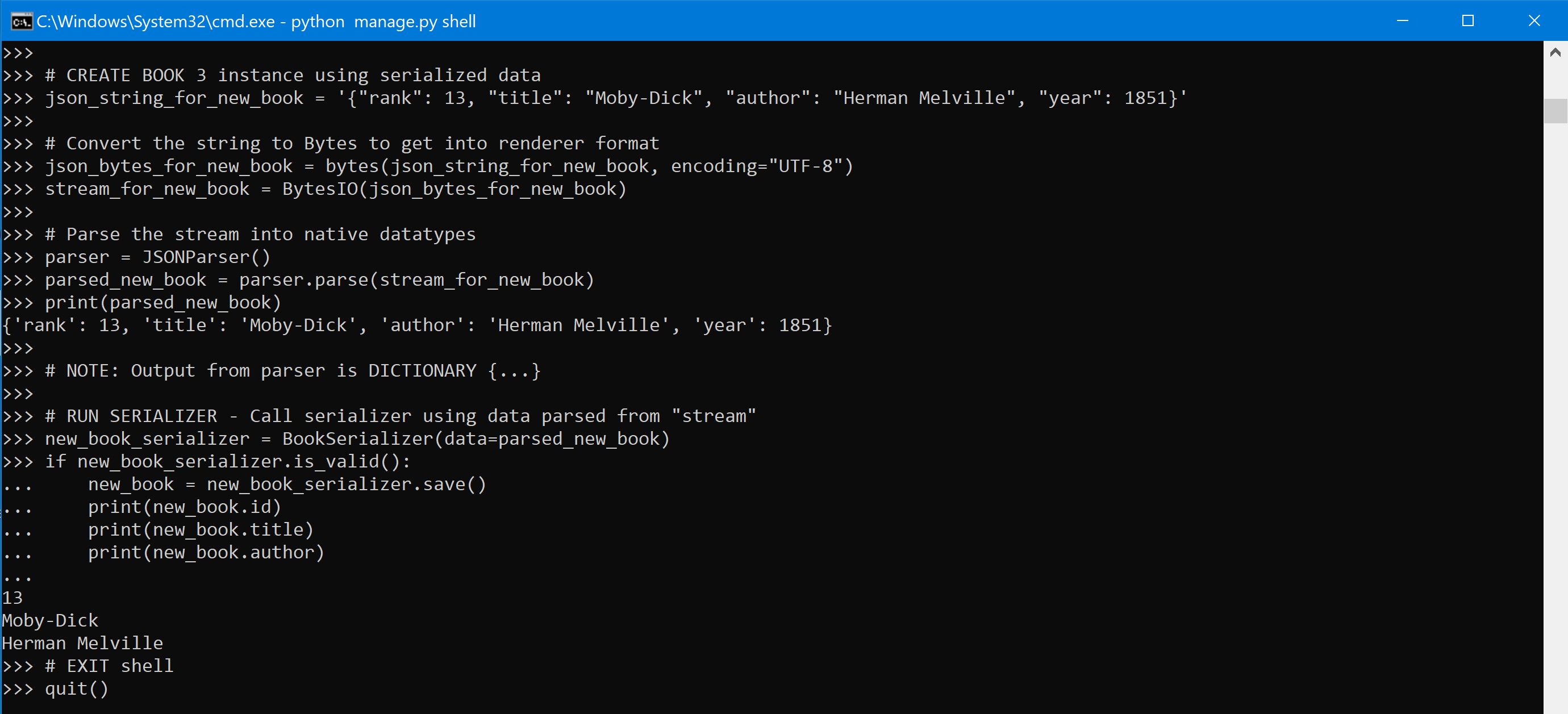
**SCREEN CAPTURE # 3 - “Create 2 new books” output [CHUNK1]**

Sample Output from CHUNK2 & CHUNK3:



**SCREEN CAPTURE # 4 - “Serialize & Deserialize” output [CHUNK2 & CHUNK3]**

Sample Output from CHUNK4:



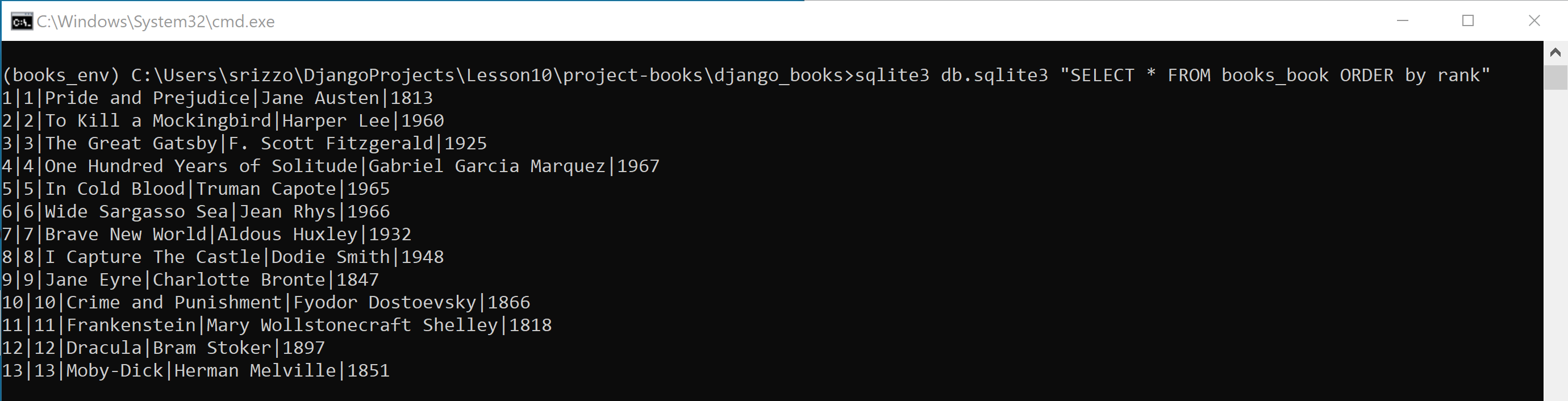
**SCREEN CAPTURE # 5 - “Create book 3” output [CHUNK4]**

After running the code in the Python interactive shell, you should have 3 new Book instances persisted in the database (for a total of 13).

### **Verification – books\_book**

Use **sqlite3** command to view all the entries in **books\_book** table:

* **Enter** the following command from the command line:
  + sqlite3 db.sqlite3 “SELECT \* FROM books\_book ORDER by rank”



**SCREEN CAPTURE # 6 - Command window output with 13 Books displayed**

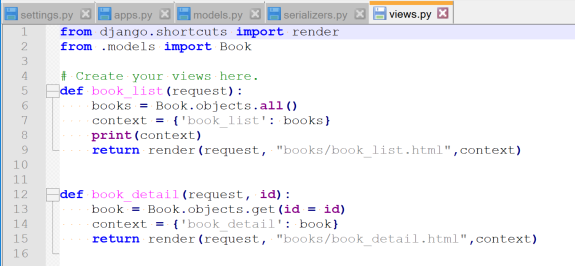
### **Writing API views**

Now that our Model and Serializer has been created & tested, let’s create Django views that will use the **BookSerializer** code to return JSON for each HTTP request our API is setup to handle.

Create **views** to use depending on the HTTP request:

* **Edit** the project\_books\django\_books\**books\views.py** file.

This file was automatically created for us when the **books** application was first created and currently contains the view definitions for your **book\_list** and **book\_detail** views.



* **Copy** the code provided in **book-api-code.txt**
  + Replace all **import** statements & copy **class** code *before* existing **def**’s

Review the contents & comments in this file. The view added to this file creates a **JSONResponse** class with 2 functions: book\_collection & book\_detail.

The **book\_collection** method handles GET and POST requests. The GET request returns all the Book instances, and the POST request creates a new Book instance.

The **book\_instance** method handles GET, PUT and DELETE requests. The GET request returns the Book instance for a specific id, the PUT request updates the Book instance for a specific id with data passed in, and the DELETE request deletes the Book instance for a specific id.

**NOTE:** Before we can use these views, we need to “map” them to a URL pattern(s)

### **Mapping API views to PATHs**

Create the **URLs** code to map specific urlpatterns to the **Views**.

* **Edit** the project\_books\django\_books\**books\urls.py** file.
* **Copy** the code provided in **book-api-code.txt** (replace all code)

### **Verify by Making HTTP requests to the Django API via the Browser**

Everything should be all set-up to use our RESTful APIs … let’s give them a try.

Start the Django development server (if not already running):

* **Enter** the following command from the command line:
  + python manage.py runserver

OR

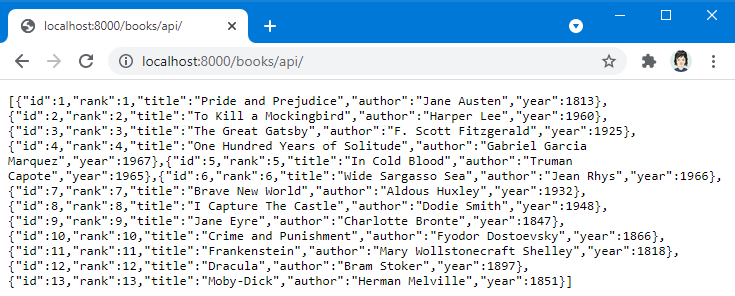
* + start python manage.py runserver *🡨 starts in new window*

The server should start without **ERROR**. If not, **STOP** and retrace your steps.

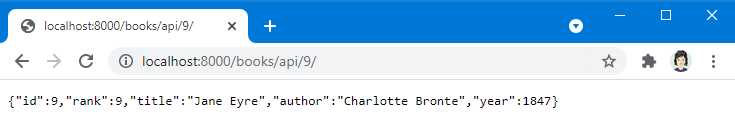
Open a browser window and:

* **Enter** the following URLs in the address window:
  + localhost:8000/books/api/
  + localhost:8000/books/api/11/

Sample Outputs:



**SCREEN CAPTURE # 7 - Browser with 13 books displayed**



**SCREEN CAPTURE # 8 - Browser with 1 Book displayed (id = 9)**

### **Verify by Making HTTP requests to the Django API via the Browser**

At the very beginning of this lab, when you installed the **djangorestframework** package you also installed a package called **httpie**.

[URL: <https://httpie.io/docs> ]

This package provides a command line HTTP client that can be useful when testing APIs & HTTP servers. Using this simple command line tool, you can verify the all the CRUD requests for your **books** application. The following section will have you test each of the CRUD operations.

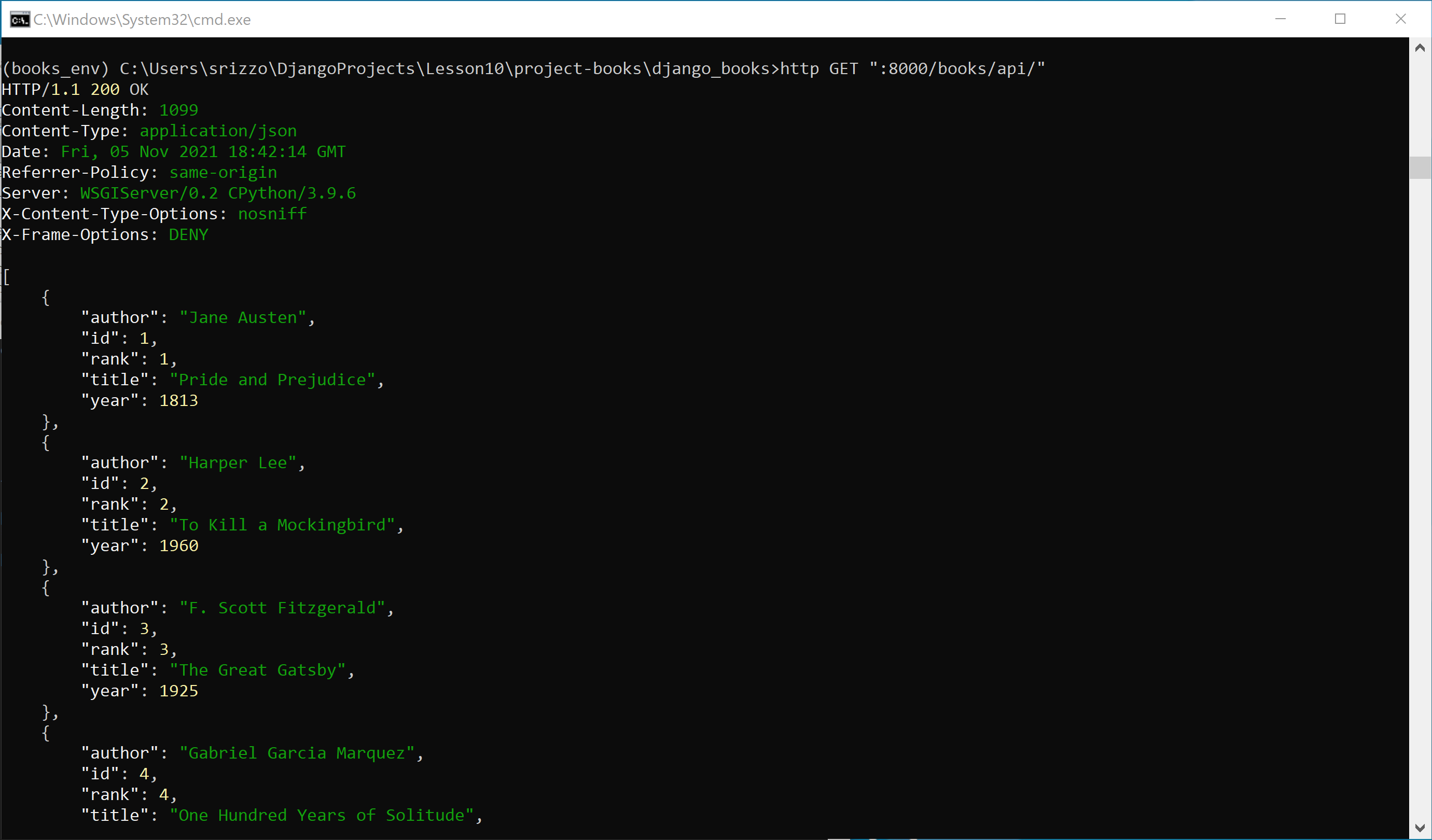
* + **C**reate ~ create a new book instance
  + **R**ead ~ read all the books or one of the books
  + **U**pdate ~ update a book
  + **D**elete ~ delete a book

#### Using HTTP GET ~ read all the books

From the command prompt window (in django\_books directory):

* **Enter** the following HTTPie command:
  + http GET “:8000/books/api/”

Sample Output (truncated to save space):



**SCREEN CAPTURE # 9 – HTTP GET results from /books/api/ using API**

**NOTE:** The command text is provided to you in file **HTTPie-code.txt.**

You can **copy** the commands provided to the command prompt to avoid typos.

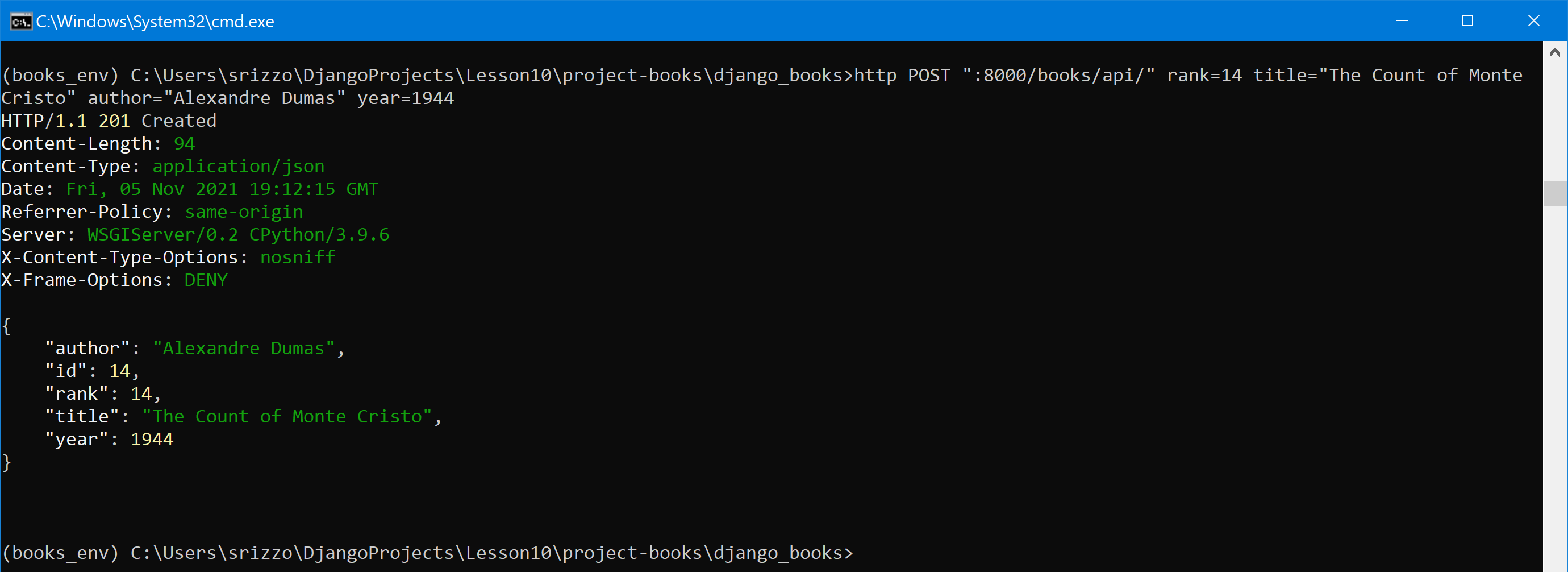
#### Using HTTP POST ~ create a new book

From the command prompt window (in django\_books directory):

* **Enter** the following HTTPie command:
  + http POST “:8000/books/api/” rank=14 title=”The Count of Monte Cristo” author=”Alexandre Dumas” year=**1944**

**YES** ~ the year value is intentionally incorrect. You will correct it shortly.

Sample Output:



**SCREEN CAPTURE # 10 – HTTP POST results from /books/api/ using API**

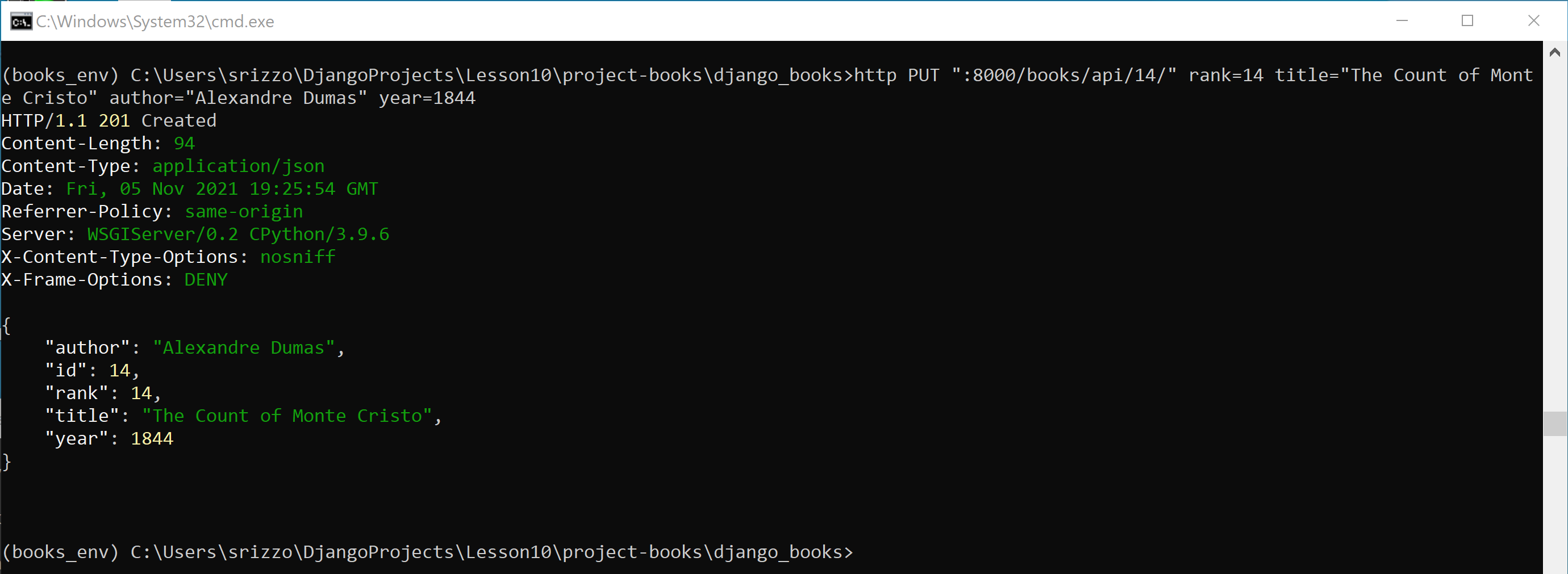
#### Using HTTP PUT ~ update a book

From the command prompt window (in django\_books directory):

* **Enter** the following HTTPie command:
  + http PUT “:8000/books/api/14/” rank=14 title=”The Count of Monte Cristo” author=”Alexandre Dumas” year=**1844**

**CORRECTION** ~ this will fix year value that was incorrect.

Sample Output:



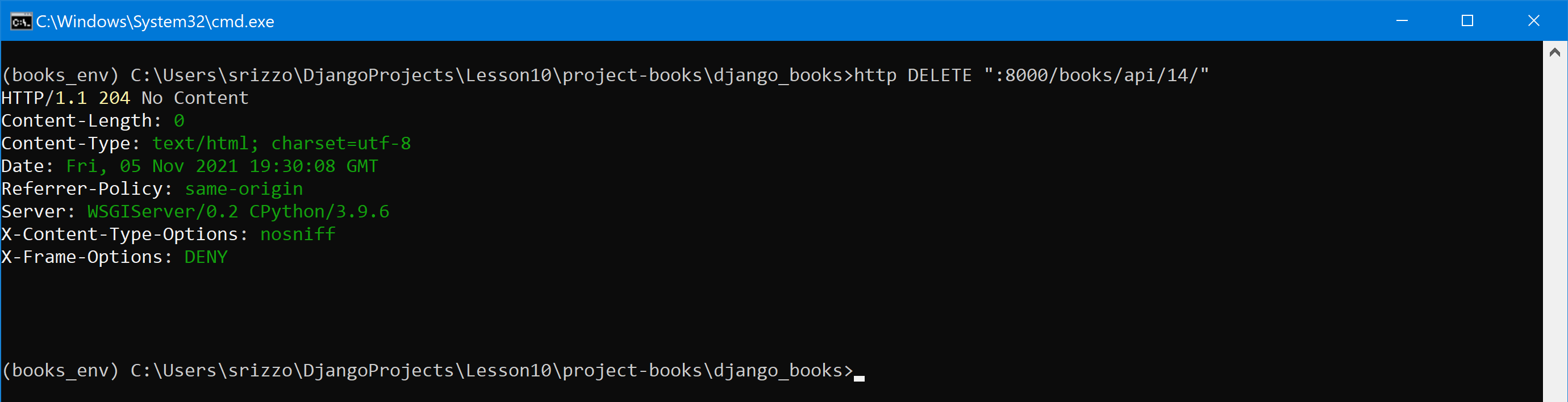
**SCREEN CAPTURE # 11 – HTTP PUT results from /books/api/14/ using API**

#### Using HTTP DELETE ~ delete a book

From the command prompt window (in django\_books directory):

* **Enter** the following HTTPie command:
  + http DELETE “:8000/books/api/14/”

Sample Output:



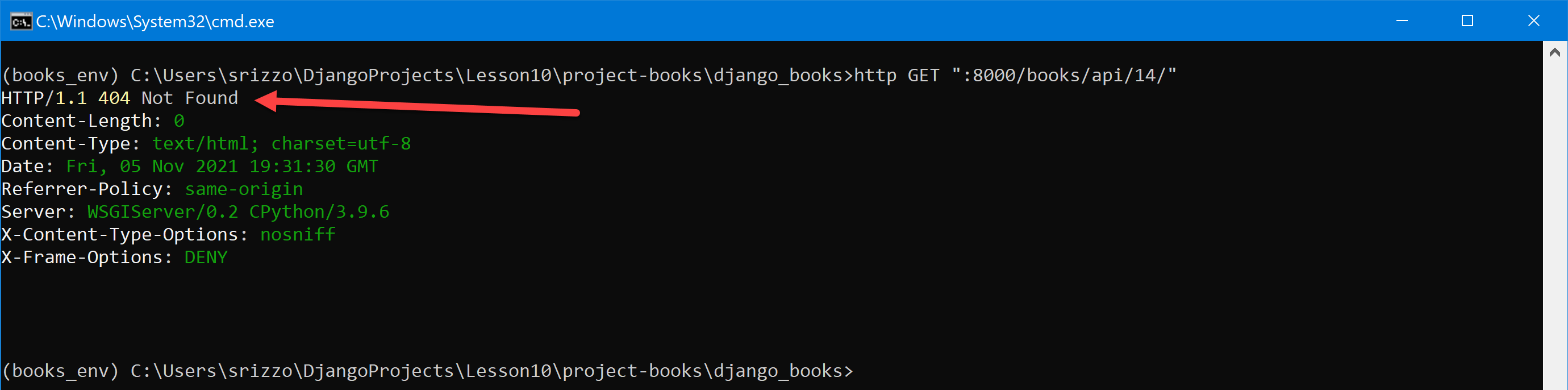
**SCREEN CAPTURE # 12 – HTTP DELETE results from /books/api/14/ using API**

#### Double-Check the HTTP DELETE ~ HTTP/1.1 404 Not Found

From the command prompt window (in django\_books directory):

* **Enter** the following HTTPie command:
  + http GET “:8000/books/api/14/”

Sample Output:



# Submitting Assignment

Once you have successfully completed the lab & gathered the **(12)** required screen captures, upload (submit) file:

* **Lab12-BookAppScreenCaptures.docx**
* **Lab12-RESTAPI.zip 🡨** zip the 2 directories (**books** & **django\_books**)

Your directory structure should look like this:

* project-books\django\_books\**books <DIR>**
* project-books\django\_books\**django\_books <DIR>**
* project-books\django\_books\**manage.py**
* project-books\django\_books\**db.sqlite3**

via Blackboard for grading.

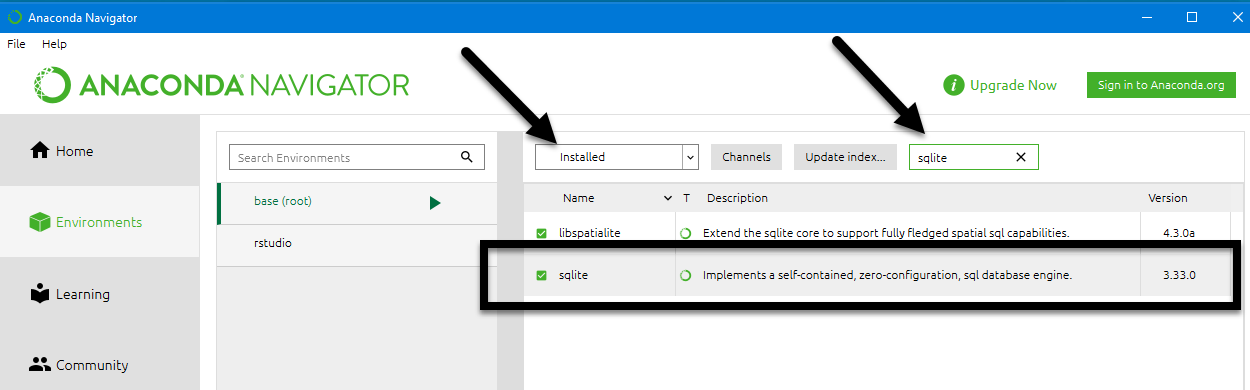
# Grading rubric

Total points = 150

* 10 points per (12) screen capture = 120
* 30 points for ZIP file = 30

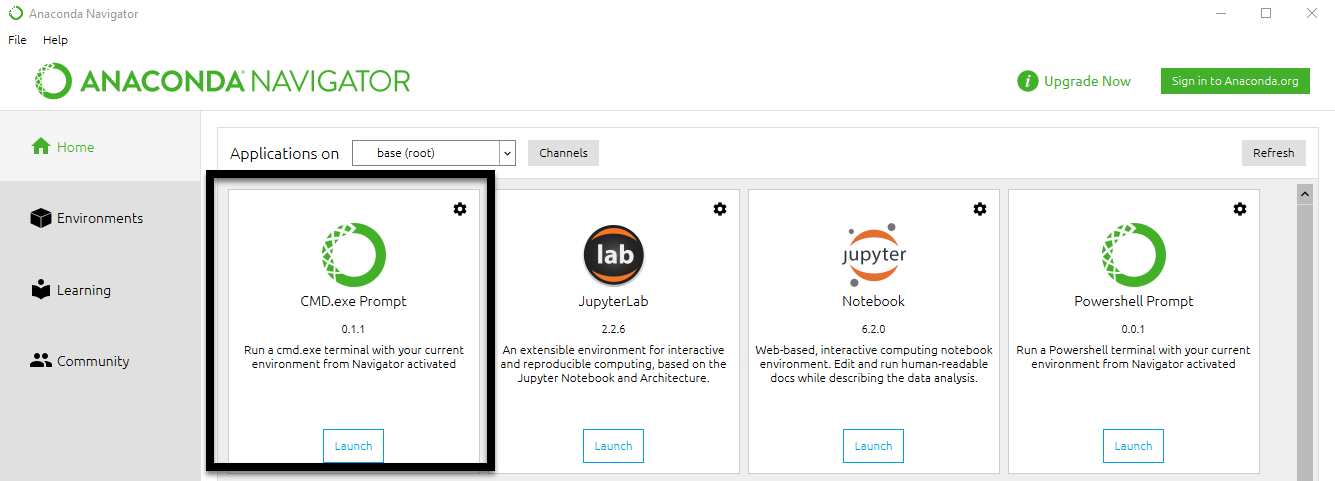
# ADDENDUM

If you cannot run **sqlite3** in your command window you probably do ***not*** have the Anaconda version of Python first in your system PATH. The sqlite3 command line tool is included with the base installation of Anaconda/Python.

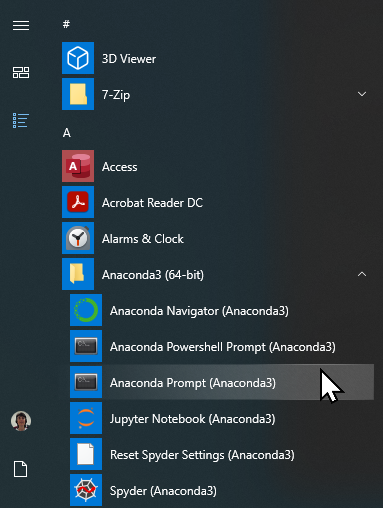


Therefore, if you cannot run **sqlite3** in your command window, you should open a command window (CMD.exe) within Anaconda – this will assure you are using the Anaconda installation environment. There are 2 ways to do this:

1. Open CMD.exe from Anaconda Navigator



1. Open Anaconda Prompt (Anaconda3) from the Start Menu [*Windows*]



Then, navigate to the required directory […\project-books\django\_books]