Lab - Explore Rest APIs with API Simulator and Postman (Instructor Version)

Instructor Note: Red font color or gray highlights indicate text that appears in the instructor copy only.

Answers: 4.5.5 Lab - Explore REST APIs with API Simulator and Postman

Objectives

Part 1: Launch the DEVASC VM

Part 2: Explore API Documentation Using the API Simulator

Part 3: Use Postman to Make API Calls to the API Simulator

Part 4: Use Python to Add 100 Books to the API Simulator

Background / Scenario

The DEVASC VM includes a School Library API simulator with API documentation and an associated database. You can use the simulator offline to explore APIs and test their functionality.

In this lab, you will learn how to use the School Library API simulator to make API calls to list, add, and delete books. Later, you will use Postman to make these same API calls.

Required Resources

- 1 PC with operating system of your choice
- Virtual Box or VMWare
- DEVASC Virtual Machine

Instructions

Part 1: Launch the DEVASC VM

If you have not already completed the **Lab - Install the Virtual Machine Lab Environment**, do so now. If you have already completed that lab, launch the DEVASC VM now.

Part 2: Explore API Documentation Using the API Simulator

To understand how to make calls to a REST API, developers typically start by studying the API documentation. The format for requests, responses, headers, and parameter for REST APIs are typically documented using the OpenAPI Specification (formerly Swagger Specification).

Step 1: Open the Chromium Web Browser.

Double-click the Chromium Web Browser icon on the desktop.

Step 2: Connect to the School Library Web Site.

If the browser does not automatically open the School Library website, in the address bar type: **library.demo.local** and press return to go there.

Step 3: Go to the API docs page.

a. The web site defaults to the **Our Books** tab and displays a list of books. In the upper right corner where it states, **Click here for API docs**, click **here** to go to the API documentation web page.

You will now see a list of APIs in the **/api/v1 Default namespace**.

b. Notice the downward arrow to the far right. Clicking anywhere on the **/api/v1** bar will minimize the API list and turn the arrow facing right. Click again on the same bar to re-display the API list.

Notice the lock to the far right of several of the APIs. The lock indicates that these APIs require a token to be used.

Step 4: List books using the GET /books API.

Click anywhere on the bar for the **GET /books** API. This API returns a list of books in the school library.

- **Parameters** - There are several optional API parameters. These can be used to filter, sort, or paginate the output. These will be referred to later in this lab.
- **Response content type** - Click **application/json** to see a list of the different types of data formats the information can be viewed. Leave the selection as **application/json**.
- **Code** - The code displays 200 by default, which indicates the API request from the sever was a success as displayed in the **Description**. (You have not sent an API request yet.)

Step 5: Use the Try it out feature in the API documentation.

One of the more powerful features of the OpenAPI Specification is the ability to test an API call to see if you constructed it correctly. You can also review the response to see if it is what you expected. You will see this same testing feature in API documentation for Cisco, MapQuest, and other organizations that use this OpenAPI Specification feature.

a. In the **GET /books** API documentation, click the **Try it out** button.

b. Notice that you now have the option to enter information for the optional parameters. Leave the parameters blank and click the **Execute** button.

In the **Responses** section you will see:

- **Curl:** The curl command you can use to access the same information for the **/books** API.
- **Request URL:** This URL is used in the API request, which can be used to request the same information using curl, Postman, and Python.
- **Code**: This is the HTTP response code. 200 indicates a successful call.
- **Response body**: List of books in JSON format.
- **Response headers**: Information about the API returned from the server.

In the **Response body** you will see a list of books in JSON format:


```
[
    {
        "id": 0,
        "title": "IP Routing Fundamentals",
        "author": "Mark A. Sportack"
    },
    {
        "id": 1,
        "title": "Python for Dummies",
        "author": "Stef Maruch Aahz Maruch"
```

```
    },
    {
        "id": 2,
        "title": "Linux for Networkers",
        "author": "Cisco Systems Inc."
    },
    {
        "id": 3,
        "title": "NetAcad: 20 Years Of Online-Learning",
        "author": "Cisco Systems Inc."
    }
]
```

Step 6: Use the curl command in a terminal window.

The **GET /books** API provides information to access the content displayed in the response body using curl. curl is a command line tool to transfer data to or from a server, using any of the supported protocols including HTTP and HTTPS.

a. Select the curl command, right-click and **Copy** it to your clipboard:

```
curl -X GET "http://library.demo.local/api/v1/books" -H "accept: application/json"
```

Open a terminal window. Right-click and **Paste** the contents from the clipboard into the terminal and press Enter. Notice this provides the same information as the library's OpenAPI interface.

```
devasc@labvm:~$ curl -X GET "http://library.demo.local/api/v1/books" -H "accept: application/json"
[
    {
        "id": 0,
        "title": "IP Routing Fundamentals",
        "author": "Mark A. Sportack"
    },
    {
        "id": 1,
        "title": "Python for Dummies",
        "author": "Stef Maruch Aahz Maruch"
    },
    {
        "id": 2,
        "title": "Linux for Networkers",
        "author": "Cisco Systems Inc."
    },
    {
        "id": 3,
        "title": "NetAcad: 20 Years Of Online-Learning",
        "author": "Cisco Systems Inc."
    }
]
devasc@labvm:~$
```

Step 7: List books with their ISBN using the GET /books API.

a. Return to the School Library API web site's GET /books API.

b. In the **Parameters** section, select the down arrow next to the i**ncludeISBN** parameter and select **true**.

c. Click **Execute**.

Notice the following changes in **Responses**:

- **Curl** now includes the parameter for ISBN.

```
curl -X GET "http://library.demo.local/api/v1/books?includeISBN=true" -H
"accept: application/json"
```

- **Request URL** now includes the parameter for ISBN.

```
http://library.demo.local/api/v1/books?includeISBN=true
```

- **Response body** has the same list of books as shown previously but now includes the book's ISBN.

To minimize the scrolling, when you are done with an API you can close that specific API window by clicking anywhere on the title bar. Now you can see all the APIs more easily.

Step 8: Get a token using the POST /loginViaBasic API.

a. Click the API **POST /loginViaBasic**.

b. Notice there are no parameters. Click **Try it out**, and then click **Execute**.

c. A **Sign in** box will prompt you for a **username** and **password**. Enter the following information and click **Sign in**:

- **Username**: cisco
- **Password**: Cisco123!

d. The token will be displayed in the **Response body**. Select the information between the quotes, right-click and **Copy** the information into your clipboard. **Your token will differ from the one shown below**.

```
{
  "token": "cisco|KZZzteQbC5iV3HKEzB7hCJ6qHQXen4rLGh72YJKeVfs"
}
```

e. Scroll up to the top of the School Library API page and click the green **Authorize** button. The **Available authorizations** dialogue box will appear.

f. Right-click and **Paste** the token after **Value** and click **Authorize**. Notice the **Name** is X-API-KEY. This information along with the token will be used later in Postman.

g. Close the **Available authorizations** dialogue box and return to the list of APIs. Notice the locks by several of the APIs have now changed. These APIs are now available for you to use.

h. Click the bar for the API **POST /loginViaBasic** to close the window.

Step 9: Add books using the POST /books API.

a. Click the API **POST /books**.

b. Notice under **Parameters** that the **payload** is required. This means that this API requires information for this parameter in the format specified by the **Parameter content type**, which is JSON.

c. Click **Try it out**.

d. Modify the **id**, **title** and **author** with the information shown below.

```
{
```

```
  "id": 4,
  "title": "IPv6 Fundamentals",
  "author": "Rick Graziani"
}
```

e. Click **Execute**.

f. Verify that the post was successful in the Server response. A **Code** of 200 means the post was a success. You should see the book you added in the **Response body** along with a new **id**. You will also see updated information for **curl** and the **Request URL**.

g. To add another book, modify the **id**, **title** and **author** with the information shown below.

```
{
  "id": 5,
  "title": "31 Days Before Your CCNA Exam",
  "author": "Allan Johnson"
}
```

h. Click **Execute**.

i. Verify that the post was successful in the **Server response**. A **Code** of 200 means the post was a success. You should see the book you added in the **Response body** along with a new **id**. You will also see updated information for **curl** and the **Request URL**.

Note: If you got a **401** code, check the **Response body** text. Most likely you received an **"error": "Invalid API key"** response. This is because you did not enter all the characters for your API key. Or possibly, you add an unnecessary space. Return to the previous step and repeat the authorization process.

j. Click the bar for the API **POST /books** to close the window.

k. You can verify the books were added to the **Our Books** page. Return to the **School Library** tab in your browser (http://library.demo.local) and refresh the page. Be careful not to close the School Library API tab. If you do, then you will need to reauthenticate.

Step 10: List books using the GET /books API.

a. Return to the **School Library API** tab in the browser. Click the **GET /books** API.

b. Click **Try it out**. If you see **Cancel** button in red, then you are already in **Try it out** mode.

c. Click **Execute**.

d. Under **Server response** in the **Response body**, you will now see the two books you added. Notice they each have a unique **id**.

```
[
  {
    "id": 0,
    "title": "IP Routing Fundamentals",
    "author": "Mark A. Sportack"
  },
  {
    "id": 1,
    "title": "Python for Dummies",
    "author": "Stef Maruch Aahz Maruch"
  },
  {
```

```
    "id": 2,
    "title": "Linux for Networkers",
    "author": "Cisco Systems Inc."
  },
  {
    "id": 3,
    "title": "NetAcad: 20 Years Of Online-Learning",
    "author": "Cisco Systems Inc."
  },
  {
    "id": 4,
    "title": "IPv6 Fundamentals",
    "author": "Rick Graziani"
  },
  {
    "id": 5,
    "title": "31 Days Before Your CCNA Exam",
    "author": "Allan Johnson"
  }
]
```


e. Click the bar for the **GET /books** API to close the window.

Step 11: List a specific book using the GET /books{id} API.

a. Click the **GET /books{id}** API. Notice this API requires the **id** as a parameter.

b. To the right of Parameters, click the **Try it out** button.

c. Under **Parameters**, enter 4 for the required id.

d. Click **Execute**. Notice the information provided by **Curl** and **Request URL**.

- **Curl** - This is the curl command to perform the same function using curl.
- **Request URL** - This is the URL that can be used to get the same information using Postman and Python.

Verify that the get was successful in the **Server response**. A **Code** of 200 means the post was a success. In the **Response body** you will see the book you requested with the **id** of 4.


```
{
  "id": 4,
  "title": "IPv6 Fundamentals",
  "author": "Rick Graziani"
}
```


e. Click the bar for the **GET /books{id}** API to close the window.

Step 12: Delete a specific book using the DELETE /books{id} API.

a. Click the **DELETE /books{id}** API. Notice this API requires the **id** as a parameter.

b. Click **Try it out**.

c. Under **Parameters**, enter 4.

d. Click **Execute**.

e. Verify that the delete was successful in the **Server response**. A **Code** of 200 means the post was a success. In the **Response body** you will see the book you deleted with the **id** of 4.


```
{
  "id": 4,
  "title": "IPv6 Fundamentals",
  "author": "Rick Graziani"
}
```


f. Click the bar for the **DELETE /books{id}** API to close the window.

Step 13: List books using the GET /books API.

a. Click the **GET /books** API.

b. Click **Try it out**. If you see **Cancel** button in red, then you are already in **Try it out** mode.

c. Click **Execute**.

d. Under **Server response** in the **Response body**, you will no longer see the book with **id** or 4.


```
[
  {
    "id": 0,
    "title": "IP Routing Fundamentals",
    "author": "Mark A. Sportack"
  },
  {
    "id": 1,
    "title": "Python for Dummies",
    "author": "Stef Maruch Aahz Maruch"
  },
  {
    "id": 2,
    "title": "Linux for Networkers",
    "author": "Cisco Systems Inc."
  },
  {
    "id": 3,
    "title": "NetAcad: 20 Years Of Online-Learning",
    "author": "Cisco Systems Inc."
  },
  {
    "id": 5,
    "title": "31 Days Before Your CCNA Exam",
    "author": "Allan Johnson"
  }
]
```


Note: Do not close the **School Library API** tab in the Chromium browser. You will use the API documentation in the next part.

Part 3: Use Postman to Make API Calls to the API Simulator

In this Part, you will use Postman to make the same API calls you made in the Student Library API documentation. Postman is a useful tool when an API developer web site is not available while providing the ability to easily save, organize, and reuse APIs.

Step 1: Open Postman.

Double-click the Postman icon on the desktop. Normally, you would sign in to Postman. However, it is not necessary to get an account and login to Postman for labs in this course.

Step 2: List the books using the GET /books API.

a. In the main window next to the **Launchpad** tab, click the plus icon "+" to create an **Untitled Request**. By default, this will be a **GET** request.

b. Click the down arrow next to **GET** to view the different API operations including GET, POST, and DELETE. Leave the selection on **GET**. Click the up arrow next to **GET** to close the list.

c. Enter request URL.

1) Return to the **School Library API** tab in Chromium and, if necessary, expand the **GET /books** API.

2) Under **Request URL**, select, right-click and **Copy** the URL to your clipboard:

`http://library.demo.local/api/v1/books`

3) Return to **Postman** and paste the URL next to GET where it states, "Enter request URL".

Note: If pasting adds a line below the URL, remove the extra line.

d. Click **Send**. To verify that the API request was a success, you will now see a response that include the **Status** code 200 OK in green. Scroll down to the **Body** section to see the response. Notice that the default is **Pretty** and **json**.


```
[
    {
        "id": 0,
        "title": "IP Routing Fundamentals",
        "author": "Mark A. Sportack"
    },
    {
        "id": 1,
        "title": "Python for Dummies",
        "author": "Stef Maruch Aahz Maruch"
    },
    {
        "id": 2,
        "title": "Linux for Networkers",
        "author": "Cisco Systems Inc."
    },
    {
        "id": 3,
        "title": "NetAcad: 20 Years Of Online-Learning",
        "author": "Cisco Systems Inc."
    },
    {
        "id": 5,
```

```
        "title": "31 Days Before Your CCNA Exam",
        "author": "Allan Johnson"
    }
]
```

Note: You can save the JSON output to a file using the **Save Response** button above the output. This is not required for this lab.

Step 3: Get a Token using the POST /loginViaBasic API.

a. In the main window, click the plus icon "+" to create a new **Untitled Request**.

b. Click the down arrow next to **GET** and select **POST**.

c. Enter request URL.

1) Return to the **School Library API** tab in Chromium and expand the **POST /loginViaBasic** API, if necessary.

2) Under **Request URL**, select, right-click and **Copy** the URL to your clipboard:

`http://library.demo.local/api/v1/loginViaBasic`

Note: If the **Request URL** is no longer showing, then you probably closed and re-opened the **School Library API** documentation page and are no longer authenticated. Click **Try it out**, then **Execute**, and then re-authenticate with username **cisco** and password **Cisco123!**.

3) Return to **Postman** and paste the URL next to POST where it states, "Enter request URL".

Note: If pasting adds a line below the URL, remove the extra line.

d. Click **Authorization**. Within this area, complete the following:

1) In the drop-down list for **Type**, choose **Basic Auth**.

2) For the **Username** and **Password** fields, fill in the following:

- **Username:** cisco
- **Password:** Cisco123!

e. Click **Send**.

f. If necessary, scroll down to the **Body** section to see your new token. **Your token will be different than the one shown here.**

```
{
    "token": "cisco|5xSUHYFDvIAoCRv0LqWVSDcjJAwWjg18vMml6u2lm1I"
}
```

Step 4: Add a book using the POST /books API.

Now you will add the *IPv6 Fundamentals* book you deleted in Part 2 when using the **Try it out** feature in the **School Library API** documentation.

a. In the main window, click the plus icon "+" to create an **Untitled Request**.

b. Click the down arrow next to **GET** and select **POST**.

c. Enter request URL.

1) Return to the **School Library API** tab in Chromium and expand the **POST /books** API.

2) Under **Request URL**, select, right-click and **Copy** the URL to your clipboard:

`http://library.demo.local/api/v1/books`

Note: If the **Request URL** is no longer showing, then you probably canceled **Try it out**. Click **Try it out**, and then **Execute** to show the **Request URL**.

3) Return to **Postman** and paste the URL next to POST where it states, "Enter request URL".

Note: If pasting adds a line below the URL, remove the extra line.

d. Click **Authorization**. Within this area, complete the following:

1) In the drop-down list for **Type**, choose **API Key**.

2) In the **Key** field, enter **X-API-KEY**.

Note: Recall that you saw **X-API-KEY** in the School Library API web page when you got a token selecting the green **Authorize** button.

3) Return to the **Post** tab in Postman and copy the token you received in Step 3. Be sure to include everything within the quotation marks. Your token will be different than the one shown here.

Example: `cisco|5xSUHYFDvIAoCRv0LqWVSDcjJAwWjg18vMml6u2lm1I`

4) Go back to the second **Post** tab in Postman. Paste the token in the **Value** field

e. In the same row with the **Authorization** tab, click **Body**. This section will allow you to choose the format of your input.

- Click the **raw** radio button.
- Click **Text** and change this option to **JSON**.

f. In the input area you will see the number 1, for "line 1". Enter the following JSON object.

```
{
  "id": 4,
  "title": "IPv6 Fundamentals",
  "author": "Rick Graziani",
  "isbn": "978 158144778"
}
```

g. Click **Send**.

h. To verify that the API request was a success, you will now see a response that include the **Status** code 200 OK in green.

Step 5: Verify the additional book with the Get /books API.

a. Return to the first GET tab. As you can see, Postman makes it easy to switch between different API calls.

b. Click **Send**.

c. To verify that the API request was a success, you will now see a response that include the **Status** code 200 OK in green.

d. Click **Body** to see the response. Notice that the default is **Pretty** and **json**.

```
[
    {
        "id": 0,
        "title": "IP Routing Fundamentals",
        "author": "Mark A. Sportack"
    },
    {
        "id": 1,
        "title": "Python for Dummies",
```

```
        "author": "Stef Maruch Aahz Maruch"
    },
    {
        "id": 2,
        "title": "Linux for Networkers",
        "author": "Cisco Systems Inc."
    },
    {
        "id": 3,
        "title": "NetAcad: 20 Years Of Online-Learning",
        "author": "Cisco Systems Inc."
    },
    {
        "id": 4,
        "title": "IPv6 Fundamentals",
        "author": "Rick Graziani"
    }
    {
        "id": 5,
        "title": "31 Days Before Your CCNA Exam",
        "author": "Allan Johnson"
    },
]
```


Step 6: Use additional parameters with the Get /books API.

a. Go to the **School Library API** web site. Scroll up to **GET /books** API and expand it, if necessary. Notice the parameters that are available:
   - **includeISBN**: Includes in the results the ISBN numbers. Default=false
   - **sortBy**: Sort results using the specified parameter. Default=id
   - **author**: Return only books by the given Author.
   - **page**: Used to specify a page number.

b. Click **Try it out**. If you see a **Cancel** button in red, then you do not need to select this button.

c. Under parameters:
   - Click **includeISBN** and select **true**
   - Click **sortBy** and select **author**

d. Click **Execute**.

e. In the **Response body** you will see the list of books now sorted by author and including the ISBNs.


```
[
    {
        "id": 5,
        "title": "31 Days Before Your CCNA Exam",
        "author": "Allan Johnson"
    },
    {
        "id": 2,
```

```
        "title": "Linux for Networkers",
        "author": "Cisco Systems Inc.",
        "isbn": "000-0000000123"
    },
    {
        "id": 3,
        "title": "NetAcad: 20 Years Of Online-Learning",
        "author": "Cisco Systems Inc.",
        "isbn": "000-0000001123"
    },
    {
        "id": 0,
        "title": "IP Routing Fundamentals",
        "author": "Mark A. Sportack",
        "isbn": "978-1578700714"
    },
    {
        "id": 4,
        "title": "IPv6 Fundamentals",
        "author": "Rick Graziani",
        "isbn": "978 1587144778"
    },
    {
        "id": 1,
        "title": "Python for Dummies",
        "author": "Stef Maruch Aahz Maruch",
        "isbn": "978-0471778646"
    }
]
```


Notice that the **Request URL** now includes the parameters. You will see this again in Postman.

`http://library.demo.local/api/v1/books?includeISBN=true&sortBy=author`

f. Return to **Postman** and go to the first API tab, GET http://library.demo.local/api/v1/books. You will now include some of the parameters from the School Library API web site.

g. Click **Params**. You will see under **Query Params** input boxes for **KEY** and **VALUE**. Enter the following information:

- Under **KEY**, enter **includeISBN** and under **Value** enter **true**

  Notice a check mark will automatically be included to the left of the value and a new row added.

- Under **KEY**, enter **sortBy** and under **Value** enter **author**

Notice that when entering these query parameters, it has updated the original URL next to the GET. This is the same **Request URL** you saw in the School Library API web site for this same API call. This is the URL Postman will be using, with these query parameters when making the API call.

`http://library.demo.local/api/v1/books?includeISBN=true&sortBy=author`

h. Click **Send**.

Notice in the **Body**, it now shows the same list of books, sorted by author and including the ISBNs that you saw in the School Library API web site.


```
[
```

```
        {
            "id": 5,
            "title": "31 Days Before Your CCNA Exam",
            "author": "Allan Johnson"
        },
        {
            "id": 2,
            "title": "Linux for Networkers",
            "author": "Cisco Systems Inc.",
            "isbn": "000-0000000123"
        },
        {
            "id": 3,
            "title": "NetAcad: 20 Years Of Online-Learning",
            "author": "Cisco Systems Inc.",
            "isbn": "000-0000001123"
        },
        {
            "id": 0,
            "title": "IP Routing Fundamentals",
            "author": "Mark A. Sportack",
            "isbn": "978-1578700714"
        },
        {
            "id": 4,
            "title": "IPv6 Fundamentals",
            "author": "Rick Graziani",
            "isbn": "978 1587144778"
        },
        {
            "id": 1,
            "title": "Python for Dummies",
            "author": "Stef Maruch Aahz Maruch",
            "isbn": "978-0471778646"
        }
    ]
```


Part 4: Use Python to Add 100 Books to the API Simulator

You could use the OpenAPI Specification Try It tool or Postman to add as many books as you want. However, you would have to add them one at a time. A better solution would be to write a program to add the books. In this Part, you will simulate the process of adding 100 books by using the Python **faker** library.

Step 1: Open Visual Studio (VS) Code and navigate to the school-library directory.

a. Open **VS Code** from the **Menu** button or by double-clicking the icon on the desktop.

b. Click **File > Open Folder...**, navigate to the **labs/devnet-src/school-library** folder, and click **OK**.

Step 2: Investigate the libraries used by add100RandomBooks.py program.

a. In VS Code **EXPLORER** pane on the left, click **add100RandomBooks.py** to open it, if necessary.

b. At the top, notice the “shebang” that sets the interpreter to Python 3 and then the three libraries that are imported.

```
#!/usr/bin/env python3

import requests
import json
from faker import Faker
```

c. You will use the Python **requests** library throughout this course. The **requests** library is required if you want to use Python to make API requests using GET, POST, DELETE and other HTTP methods.

d. Faker is a Python library that generates 'fake' data for you. This program uses the Python **faker** library to generate random book titles, authors, and ISBNs. You can search the internet for more information on **faker** library. However, complete the following steps to see all the 252 methods for the faker library.

1) Open a terminal window and start Python 3.

2) From **faker** import the **Faker()** module.

3) Assign the **Faker()** module to **fake**.

4) To see all the methods, enter **fake.** and then press the tab key twice. Notice the method **fake.name()**, which you will use in the next step. In the next step and later in this lab, you will also use the three highlighted methods (prefaced with **fake.**).: **catch_phrase()**, **isbn13()**, and **name()**.

```
devasc@labvm:~/labs/devnet-src/school-library$ python3
Python 3.8.2 (default, Apr 27 2020, 15:53:34)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from faker import Faker
>>> fake = Faker()
>>> fake.<press Tab key twice – be sure you include the period>
Display all 252 possibilities? (y or n)
fake.add_provider(              fake.future_datetime(           fake.pyfloat(
fake.address(                   fake.generator_attrs            fake.pyint(
fake.am_pm(                     fake.get_formatter(             fake.pyiterable(
<output omitted>
fake.catch_phrase(              fake.ipv4_public(               fake.random_element(
fake.chrome(                    fake.isbn10(                    fake.random_int(
fake.city(                      fake.isbn13(                    fake.random_letter(
<output omitted>
fake.date_time_ad(              fake.msisdn(                    fake.texts(
fake.date_time_between(         fake.name(                      fake.time(
<output omitted>
fake.future_date(               fake.pydict(                    fake.zipcode_plus4(
>>>
```

Step 3: Practice generating random data using the faker library.

a. Enter the following to generate a fake name. Your output will be a different fake name each time you execute the command.

```
>>> print('My name is {}.'.format(fake.name()))
My name is Katherine Ross.
>>>
```

b. Using the three highlighted messages in Step 2d above, enter the command that would print the following fake output.

```
>>> print('My name is {} and I wrote "{}" (ISBN
{}).'.format(fake.name(),fake.catch_phrase(),fake.isbn13()))
My name is Gary Castaneda and I wrote "Organic incremental neural-net" (ISBN 978-0-
669-01935-3).
>>>
```

c. Later in the program, a loop is used to iterate through these three methods to create entries for the School Library. Enter the following to generate 10 random names. After the "…" you will need to press return a second time.

```
>>> for i in range(10):
...     print(fake.name())
...
Kevin Moyer
Mr. Christopher Green MD
Spencer Jensen
Whitney Guzman
Nicole Scott
Tammy Lewis
Craig Edwards
Michael Diaz
Ryan Mccoy
Terry Rocha
>>>
```

d. Quit the Python interpreter when done investigating the **faker** library.

```
>>> quit()
devasc@labvm:~/labs/devnet-src/school-library$
```

Step 4: Review the function variables.

The two functions in the program use three variables to get an authorization token from the School Library API service.

```
APIHOST = "http://library.demo.local"
LOGIN = "cisco"
PASSWORD = "Cisco123!"
```

Step 5: Review the getAuthToken function.

a. The **getAuthToken** function uses three variables to build a request. The **r** variable invokes the POST method for the **loginViaBasic** API and stores the token value if the call is successful. Notice the use of an f-string to build the request URL.

```
def getAuthToken():
    authCreds = (LOGIN, PASSWORD)
    r = requests.post(
        f"{APIHOST}/api/v1/loginViaBasic",
        auth = authCreds
    )
```

b. If the call is unsuccessful (HTTP status code is not equal to 200), an exception is raised and printed to the terminal window. Again, notice the use of an f-string to build the exception message. You can test the exception code by changing the one of the variables in Step 4.

```
    if r.status_code == 200:
        return r.json()["token"]
    else:
        raise Exception(f"Status code {r.status_code} and text {r.text}, while trying
to Auth.")
```

Step 6: Review the addBook function.

a. Similar to the **addAuthToken** function, the **addBook** function uses the three variables shown in Step 4 to build a request. The **r** variable invokes the POST method for the **books** API. The data comes from a variable called **book**, which is specified in the final part of the program.

```
def addBook(book, apiKey):
    r = requests.post(
        f"{APIHOST}/api/v1/books",
        headers = {
            "Content-type": "application/json",
            "X-API-Key": apiKey
            },
        data = json.dumps(book)
    )
```

b. If the call is unsuccessful, an exception is raised and printed to the terminal window. You can test it by changing the one of the variables in Step 4.

```
    if r.status_code == 200:
        print(f"Book {book} added.")
    else:
        raise Exception(f"Error code {r.status_code} and text {r.text}, while trying
to add book {book}.")
```

Step 7: Review the code that invokes the two functions.

a. The **addAuthToken** function is invoked and the results are stored in the variable apiKey.

```
apiKey = getAuthToken()
```

b. The **Faker()** module is set to a variable named **fake**. A **for** loop then iterates 100 times. The **i** variable is used later in the loop to set the value for the **id** key for each new book from 4 up to and not including 104.

Note: If you want to keep the two previous books added previously in this lab, change **range** to **(6, 106)**.

Instructor Note for LA reviewers: Your version of the script may have the range from 4 to 105, which will add 101 books. Change the range and save the script to only add 100 books. This will be updated in the GA release of the DEVASC VM.

```
fake = Faker()
for i in range(4, 104):
```

c. Next, three variables hold the value of methods invoked from the **Faker()** module: **catch_phrase()**, **name()**, and **isbn13()**.

```
    fakeTitle = fake.catch_phrase()
    fakeAuthor = fake.name()
    fakeISBN = fake.isbn13()
```

d. Recall that the **payload** parameter for the **books** API requires JSON in the following format:

```
{
  "id": 0,
  "title": "string",
  "author": "string"
}
```

The **book** variable is built using the three required keys for the payload parameter and values from the three fake variables.

```
book = {"id":i, "title": fakeTitle, "author": fakeAuthor, "isbn": fakeISBN}
```

e. Finally, the **addBook** function is called passing the **book** and **apiKey** variables. Because **addBook** is part of the loop, it will be called 101 times, one time each for book ID 4 through 105.

Step 8: Run and verify the add100RandomBooks.py program.

a. Enter the Python 3 command to run the **add100RandomBooks.py** program. You should get output similar to the following although your book titles and ISBN numbers will be different fake values.

```
devasc@labvm:~/labs/devnet-src/school-library$ python3 add100RandomBooks.py
Book {'id': 4, 'title': 'Assimilated client-server frame', 'author': 'Chelsea Mitchell', 'isbn': '978-0-411-83123-3'} added.
Book {'id': 5, 'title': 'Adaptive tangible conglomeration', 'author': 'Edward Ryan', 'isbn': '978-1-64406-014-8'} added.
<output omitted>
Book {'id': 103, 'title': 'Fundamental uniform data-warehouse', 'author': 'Dennis David', 'isbn': '978-1-68465-896-1'} added.
Book {'id': 104, 'title': 'Organic 4thgeneration functionalities', 'author': 'Nicole Gilbert', 'isbn': '978-0-13-176202-2'} added.
devasc@labvm:~/labs/devnet-src/school-library$
```

b. Return to the Chromium browser and refresh the **http://library.demo.local/** webpage. You should now see your 100 new books added.

Note: If you got to the API documentation page instead of the main page (**http://library.demo.local/api/v1/docs**) and use **Try It out**, you will only get a list of the first 10 books. You can enter a value from 2 to 10 the **page** parameter to see the other books.

How would you add another 100 books?

Change the for loop to have a range of 104 to 204.