Lab: Calling REST APIs with Python

JR Rickerson

Contents

Lab Guide

Read this lab like a book, all the text is there for a reason!

→ **Denotes an action you must take**
<> Denotes a placholder you should replace with a value (Ex. <IP> becomes `127.0.0.1`)

Use your favorite editor to edit files within the console, such as vim, nano, or emacs.

```
# Boxes of text like this denote command and file contents you
# should add
```

```
Boxes of text like this denote example output you should expect to see
```

Task 0: Explore the requests Python package

Step 1: Create a new working area

→

```
mkdir ~/astronomy-lab
cd ~/astronomy-lab
git init
```

Step 2: Install requests

→

```
pip install --user requests
```

Step 3: Send a request

→ **Start the Python interpreter**

```
python3
```

→ **Call to list all the events occuring in Github**

```
import requests
response = requests.get("https://api.github.com/events")
```

→ **Call to list all the events occuring in Github**

```
print(response.text)
```

As you can see, there are a ton of things going on Github!

You can see what else is available on the Response object from the **requests** library by reviewing the requests library documentation.

→ **View the status code**

```
print(response.status_code)
```

You should see “200” if your request worked correctly.

→ **Let’s grab a dictionary version of the response body (decoded from JSON)**

```
events = response.json()
```

This response is a list of “events” going on in Github. We can see the type of event that last occurred by outputting information from the first element in this list.

→ **View the most recent event**

```
print(events[0])
```

→ **See the type of event**

```
print(events[0]["type"])
```

→ **Who caused it?**

```
print(events[0]["actor"]["display_login"])
```

You have now called an API and parsed it’s JSON output. Note that many APIs have VERY deeply nested output, so it’s best to have the documentation for that API available to refer to to find the data you want to extract!

→ **Type Ctrl-D or quit() to leave the interpreter**

Task 1: Write a script to find postion of astronomical bodies

Here we will take an existing script skeleton and implement it piece by piece. Commit your code as you get things working so you save your work!

Step 1: Copy the script below

→ **solar.py**

```
#!/usr/bin/env python3
import requests
import datetime


ASTRONOMYAPI_ID="<your api id>"
ASTRONOMYAPI_SECRET="<your api secret key>"


def get_observer_location():
    """Returns the longitude and latitude for the location of this machine.

    Returns:
    str: latitude
    str: longitude"""

    # NOTE: Replace with your real return values!
    return None, None

def get_sun_position(latitude, longitude):
    """Returns the current position of the sun in the sky at the specified location

    Parameters:
    latitude (str)
    longitude (str)

    Returns:
    float: azimuth
    float: altitude
    """

    # NOTE: Replace with your real return values!
    return 0, 0

def print_position(azimuth, altitude):
    """Prints the position of the sun in the sky using the supplied coordinates

    Parameters:
    azimuth (float)
    altitude (float)"""

    print("The Sun is currently at:")


if __name__ == "__main__":
    latitude, longitude = get_location()
    azimuth, altitude = get_sun_position()
    print_position(azimuth, altitude)
```

→ **Run the app, it should output nothing, but with no errors**

```
python3 solar.py
```

Step 2: Implement the `get_location()` function

The site https://ip-api.com has a simple-to-use API that will return a JSON response containing the latitude and longitude information for a given IP address, or from the source IP address if another is not specified. If you visit the site in your browser you can see the sample response it provides for your machine. Look over the API Documentation to see the exact URL you will need to use in your GET request to get your current geolocation based on your IP address. Note this location will be different if you do it from a local machine or your AWS Workspace! Pay special attention to the fact that the public endpoint for ip-api.com does NOT use HTTPS!

→ **Use the ip-api.com to implement the `get_location()` function in your script.** I suggest you test as you go. You can do this easily via print() calls to output variable values and verify you extract the correct data.

Step 3: Implement the `get_sun_position()` function

Another API, https://astronomyapi.com, can return positions of the bodies in the solar system based on your current location and time of day. Using an existing API like this gives us access to useful information quickly, without having to look it up more manually and without having to write the calculations for the bodies ourselves.

You may have noticed two variables in the skeleton script, **ASTRONOMYAPI_ID** and **ASTRONOMYAPI_SECRET**. These must be a valid API Key and Secret given to us by the Astronomy API site to use to authenticate ourselves. You can sign up for your own account by going to https://astronomyapi.com and registering for a free account. After logging into your account, choose to "Create an Application." Give the application whatever name and Origin URL you like (the Origin URL only applies if you are accessing from front-end web applications in Javascript).

You may also want to read up on how the Horizontal Coordinate System works for astronomical bodies, to understand the values you are working with.

Once you have your Application ID and Secret, fill in the appropriate values in your script. Figure out the correct API endpoint to use to get the Sun's position at the current time.

→ **Implement the `get_sun_position()` given the location data from `get_location()`**

NOTES:

- You will need to get the current time from the Python **datetime** module and format it correctly to match what the Astronomy API expects.
- The Astronomy API uses HTTP Basic Authentication, which **requests** supports out of the box. Follow the examples in the requests Authentication docs and use your API ID for the username, and API Secret for the password.
- You will also need to use some optional parameters of `requests.get()` to pass the querystring parameters that the Astronomy API requires. Review the requests.get documentation for the correct parameters names and data types.

Step 4: Implement the `print_position()` function

This function should simply print the information we got from the API in a human-readable way. This is our "User Interface."

→ **Implement the function**

→ **Running the script should now output something like:**

```
The Sun is currently at: 0.125 degrees azimuth, 90.321 degrees altitude
```

BONUS 1: Output additional info about the Sun

→ **Update the script to output more information, similar to the following:**

```
From Atlanta, GA at 8:34 AM on March 25, 2021:

Sun:
    Distance from Earth: 149,597,900 km
    Magnitude: -26.74
    Position:
        Azimuth: 45 deg 12 min 47 sec
        Altitude: 10 deg 15 min 22 sec
```

BONUS 2: Parameterize your script

Our app has limited value. Of course it's helpful to see the Sun's position currently, but what if we wanted to break out our telescopes and location another planet to observe? We may want to know where, for instance, Mars would be on particular date and time from our location, or maybe even a nearby location. If the parameters are not provided by the user to our script, we'll simply use the automatically detected defaults we've been using.

We can use the standard library module argparse to have our script accept parameters (or use the docopt package if you want to get advanced) and the same API calls we've already made to get other solar system bodies' information. Pay special attention to the date / time formats that the API expects! Hint: The Linux command `date` we studied has easy options to generate date / time stamps in a variety of standard formats for easy testing.

→ **Make these commands work. If all is working, you should get different results for different parameters**

```
python3 solar.py --at 2021-04-01T20:00:00 mars
python3 solar.py
python3 solar.py --from lat=37.123,long=-87.125 saturn
```

BONUS 3: Error Handling

What happens if you pass in an invalid date or time to your script? What happens if your current location cannot be determined, or the APIs you're using are unavailable? Add some checks so that your application outputs a useful error message to the user, rather than just crashing. You may want to ensure that your API requests are returning an HTTP 200 status code.

BONUS 4: Advanced API Usage

In actual production applications, we rarely get the information we need from a single, simple API call. Instead, we typically must make several API calls and then combine the results into something meaningful to us, or derive other values via calculations. We must occasionally deal with things like pagination, rate limiting, etc. that can make these tasks even more difficult. Consider the APIs calls your script currently uses, and think about what you might need to do to write another script or additional functions that would answer the following questions.

1. The naked eye can be used to observe the sky, but only up to a magnitude of between +4.5 and +6.5, depending on light pollution, etc. Given a magnitude value, how could you write a function that could query the Astronomy API to find all the visible bodies in the solar system you could see at a given time and location? (NOTE: For magnitude, smaller values are brighter, and larger values are dimmer. See Apparent Magnitude)
2. Using the Astronomy API, figure out when Sunrise and Sunset are for your location on a particular date, as accurately as possible. Remember that the Altitude represents the degrees above the horizon (you can ignore mountains and hills). Check your results against other data sources to see how well you did!

3. NASA launches missions to Mars (like Perseverence) based on “launch windows” when Mars and Earth’s orbit cause the planets to come closest together. Given the Astronomy API provides us the distance from Earth, but allows a maximum span of 30 days in a single API call, how would you write a function to use the API to find the next date when Mars is nearest to Earth?