Python API tutorial - An Introduction to using APIs

Vik Paruchuri | 08 SEP 2015 in tutorials

Application Program Interfaces, or APIs, are commonly used to retrieve data from remote websites. Sites like Reddit, Twitter, and Facebook all offer certain data through their APIs. To use an API, you make a request to a remote web server, and retrieve the data you need.

But why use an API instead of a static dataset you can download? APIs are useful in the following cases:

- The data is changing quickly. An example of this is stock price data. It doesn't really make sense to regenerate a dataset and download it every minute – this will take a lot of bandwidth, and be pretty slow.
- You want a small piece of a much larger set of data. Reddit comments are one example. What if you want to just pull your own comments on Reddit? It doesn't make much sense to download the entire Reddit database, then filter just your own comments.
- There is repeated computation involved. Spotify has an API that can tell you the genre of a piece of music. You could theoretically create your own classifier, and use it to categorize music, but you'll never have as much data as Spotify does.

In cases like the ones above, an API is the right solution. In this blog post, we'll be querying a simple API to retrieve data about the International Space Station (ISS). Using an API will save us time and effort over doing all the computation ourselves.

The International Space Station.

API Requests

APIs are hosted on web servers. When you type `www.google.com` in your browser's address bar, your computer is actually asking the `www.google.com` server for a webpage, which it then returns to your browser.

APIs work much the same way, except instead of your web browser asking for a webpage, your program asks for data. This data is usually returned in JSON format (for more on this, checkout our tutorial on working with JSON data).

In order to get the data, we make a request to a webserver. The server then replies with our data. In Python, we'll use the requests library to do this. In this Python API tutorial we'll be using Python 3.4 for all of our examples.

Type of requests

There are many different types of requests. The most commonly used one, a *GET* request, is used to retrieve data.

We can use a simple *GET* request to retrieve information from the OpenNotify API.

OpenNotify has several API endpoints. An endpoint is a server route that is used to retrieve different data from the API. For example, the `/comments` endpoint on the Reddit API might retrieve information about comments, whereas the `/users` endpoint might retrieve data about users. To access them, you would add the endpoint to the *base url* of the API.

The first endpoint we'll look at on OpenNotify is the `iss-now.json` endpoint. This endpoint gets the current latitude and longitude of the International Space Station. As you can see, retrieving this data isn't a great fit for a dataset, because it involves some calculation on the server, and changes quickly.

You can see a listing of all the endpoints on OpenNotify here.

The *base url* for the OpenNotify API is `http://api.open-notify.org`, so we'll add this to the beginning of all of our endpoints.

```
# Make a get request to get the latest position of the interna
response = requests.get("http://api.open-notify.org/iss-now.js

# Print the status code of the response.
print(response.status_code)
```

```
200
```

Status codes

The request we just made had a *status code* of `200`. Status codes are returned with every request that is made to a web server. Status codes indicate information about what happened with a request. Here are some codes that are relevant to *GET* requests:

- `200` – everything went okay, and the result has been returned (if any)
- `301` – the server is redirecting you to a different endpoint. This can happen when a company switches domain names, or an endpoint name is changed.
- `401` – the server thinks you’re not authenticated. This happens when you don’t send the right credentials to access an API (we’ll talk about authentication in a later post).
- `400` – the server thinks you made a bad request. This can happen when you don’t send along the right data, among other things.
- `403` – the resource you’re trying to access is forbidden – you don’t have the right permissions to see it.
- `404` – the resource you tried to access wasn’t found on the server.

We’ll now make a GET request to `http://api.open-notify.org/iss-pass`, an endpoint that doesn’t exist, per the API documentation.

```
response = requests.get("http://api.open-notify.org/iss-pass"
print(response.status_code)
```

```
404
```

Hitting the right endpoint

`iss-pass` wasn’t a valid endpoint, so we got a `404` status code in response. We forgot to add `.json` at the end, as the API documentation states.

We’ll now make a GET request to `http://api.open-notify.org/iss-pass.json`.

```
response = requests.get("http://api.open-notify.org/iss-pass.j
print(response.status_code)
```

```
400
```

Query parameters

You'll see that in the last example, we got a `400` status code, which indicates a bad request. If you look at the documentation for the OpenNotify API, we see that the ISS Pass endpoint requires two *parameters*.

The ISS Pass endpoint returns when the ISS will next pass over a given location on earth. In order to compute this, we need to pass the coordinates of the location to the API. We do this by passing two parameters – latitude and longitude.

We can do this by adding an optional keyword argument, `params`, to our request. In this case, there are two parameters we need to pass:

- `lat` – The latitude of the location we want.
- `lon` – The longitude of the location we want.

We can make a dictionary with these parameters, and then pass them into the `requests.get` function.

We can also do the same thing directly by adding the query parameters to the url, like this: `http://api.open-notify.org/iss-pass.json?lat=40.71&lon=-74`.

It's almost always preferable to setup the parameters as a dictionary, because `requests` takes care of some things that come up, like properly formatting the query parameters.

We’ll make a request using the coordinates of New York City, and see what response we get.

```
# Set up the parameters we want to pass to the API.
# This is the latitude and longitude of New York City.
parameters = {"lat": 40.71, "lon": -74}

# Make a get request with the parameters.
response = requests.get("http://api.open-notify.org/iss-pass.

# Print the content of the response (the data the server retu
print(response.content)

# This gets the same data as the command above
response = requests.get("http://api.open-notify.org/iss-pass.
print(response.content)
```

```
b'{\n  "message": "success", \n  "request": {\n    "altitude"

b'{\n  "message": "success", \n  "request": {\n    "altitude"
```

The ISS over New York.

Working with JSON data

You may have noticed that the content of the response earlier was a `string`(although it was shown as a `bytes` object, we can easily convert the content to a string using `response.content.decode("utf-8")`).

Strings are the way that we pass information back and forth to APIs, but it's hard to get the information we want out of them. How do we know how to decode the string that we get back and work with it in Python? How do we figure out the `altitude` of the ISS from the string response?

Luckily, there's a format called JavaScript Object Notation (JSON). JSON is a way to encode data structures like lists and dictionaries to strings that ensures that they are easily readable by machines. JSON is the primary format in which data is passed back and forth to APIs, and most API servers will send their responses in JSON format.

Python has great JSON support, with the `json` package. The `json` package is part of the standard library, so we don't have to install anything to use it. We can both convert *lists* and *dictionaries* to JSON, and convert strings to *lists* and *dictionaries*. In the case of our ISS Pass data, it is a dictionary encoded to a string in JSON format.

The json library has two main methods:

- `dumps` – Takes in a Python object, and converts it to a string.
- `loads` – Takes a JSON string, and converts it to a Python object.

```python
# Make a list of fast food chains.
best_food_chains = ["Taco Bell", "Shake Shack", "Chipotle"]

# This is a list.
print(type(best_food_chains))

# Import the json library
import json

# Use json.dumps to convert best_food_chains to a string.
best_food_chains_string = json.dumps(best_food_chains)

# We've successfully converted our list to a string.
print(type(best_food_chains_string))

# Convert best_food_chains_string back into a list
print(type(json.loads(best_food_chains_string)))

# Make a dictionary
fast_food_franchise = {
    "Subway": 24722,
    "McDonalds": 14098,
    "Starbucks": 10821,
    "Pizza Hut": 7600
}

# We can also dump a dictionary to a string and load it.
fast_food_franchise_string = json.dumps(fast_food_franchise)
print(type(fast_food_franchise_string))
```

```
<class 'list'>
<class 'str'>
```

```
<class 'list'>
<class 'str'>
```

Getting JSON from an API request

You can get the content of a response as a python object by using the `.json()` method on the response.

```
# Make the same request we did earlier, but with the coordinat
parameters = {"lat": 37.78, "lon": -122.41}
response = requests.get("http://api.open-notify.org/iss-pass.j

# Get the response data as a python object.  Verify that it's
data = response.json()
print(type(data))
print(data)
```

```
<class 'dict'>
{'response': [{'risetime': 1441456672, 'duration': 369}, {'ris
```

Content type

The server doesn't just send a status code and the data when it generates a response. It also sends metadata containing information on how the data was generated and how to decode it. This is stored in the *response headers*. In Python, we can access this with the `headers` property of a response object.

The headers will be shown as a dictionary. Within the headers, `content-type` is the most important key for now. It tells us the format of the response, and how to decode it. For the OpenNotify API, the format is JSON, which is why we could decode it with the `json` package earlier.

```
# Headers is a dictionary
print(response.headers)

# Get the content-type from the dictionary.
print(response.headers["content-type"])
```

```
{'server': 'gunicorn/19.3.0', 'content-length': '520', 'conte
```

Finding the number of people in space

OpenNotify has one more API endpoint, `astros.json`. It tells you how many people are currently in space. The format of the responses can be found here.

```
# Get the response from the API endpoint.
response = requests.get("http://api.open-notify.org/astros.js
data = response.json()
```

```
# 9 people are currently in space.
print(data["number"])
print(data)
```

```
9

{'number': 9, 'people': [{'name': 'Gennady Padalka', 'craft':
```

Python API tutorial: Next steps

Now you've completed our Python API tutorial, you now should be able to access a simple API and make `get` requests. There are a few other types of `requests`, which you can learn more about, along with working with API authentication, in our dataquest APIs and scraping course.

Other recommended next steps are reading the requests documentation, and working with the Reddit API. There's a package called PRAW that makes working with the Reddit API easier in Python, but it's recommended to just use `requests` at first to learn how everything works.

Image Credit: Web API by Jonathan Coutiño from the Noun Project.