Web Applications and Services

Python Requests

✓ **Reading:** The Python Requests Library
10 min

✓ **Reading:** Useful Operations for Python Requests
10 min

Reading: HTTP GET and POST Methods
10 min

Reading: What is Django?
10 min

Module 2 Project

HTTP GET and POST Methods

HTTP supports several ***HTTP methods***, like GET, POST, PUT, and DELETE. We're going to spend time on the two most common HTTP requests: GET and POST.

The ***HTTP GET method***, of course, retrieves or ***gets*** the resource specified in the URL. By sending a GET request to the web server, you're asking for the server to GET the resource for you. When you're browsing the web, most of what you're doing is using your web browser to issue a whole bunch of GET requests for the text, images, videos, and so forth that your browser will display to you.

A GET request can have ***parameters***. Have you ever seen a URL that looked like this?

```
https://example.com/path/to/api/cat_pictures?search=grey+kitten&max_results=15
```

The question mark separates the URL resource from the resource's parameters. These parameters are one or more key-value pairs, formatted as a ***query string***. In the example above, the **search** parameter is set to "grey+kitten", and the **max_results** parameter is set to 15.

But you don't have to write your own code to create an URL like that one. With requests.get(), you can provide a dictionary of parameters, and the Requests module will construct the correct URL for you!

```
>>> p = {"search": "grey kitten",
...      "max_results": 15}
>>> response = requests.get("https://example.com/path/to/api", params=p)
>>> response.request.url
'https://example.com/path/to/api?search=grey+kitten&max_results=15'

```

You might notice that using parameters in Requests is yet another form of data serialization. Query strings are handy when we want to send small bits of information, but as our data becomes more complex, it can get hard to represent it using query strings.

An alternative in that case is using the ***HTTP POST method***. This method sends, or ***posts***, data to a web service. Whenever you fill a web form and press a button to submit, you're using the POST method to send that data back to the web server. This method tends to be used when there's a bunch of data to transmit.

In our scripts, a POST request looks very similar to a GET request. Instead of setting the **params** attribute, which gets turned into a query string and appended to the URL, we use the **data** attribute, which contains the data that will be sent as part of the POST request.

```
>>> p = {"description": "white kitten",
...      "name": "Snowball",
...      "age_months": 6}
>>> response = requests.post("https://example.com/path/to/api", data=p)
```

Let's check out the generated URL for this request:

```
>>> response.request.url
'https://example.com/path/to/api'

```

See how much simpler the URL is on this POST now? Where did all of the parameters go? They're part of the ***body*** of the HTTP message. We can see them by checking out the **body** attribute.

```
>>> response.request.body
'description=white+kitten&name=Snowball&age_months=6'

```

Ah, ha! There they are!

So, if we need to send and receive data from a web service, we can turn our data into dictionaries and then pass that as the **data** attribute of a POST request.

Today, it's super common to send and receive data specifically in JSON format, so the Requests module can do the conversion directly for us, using the **json** parameter.

```
>>> response = requests.post("https://example.com/path/to/api", json=p)
>>> response.request.url
'https://example.com/path/to/api'
>>> response.request.body
b'{"description": "white kitten", "name": "Snowball", "age_months": 6}'
```

And that's it for our brief introduction to the Requests module. If you want to learn more, feel free to work through the Requests Quickstart.

In the project at the end of this module, you'll use the Requests module to interact with a web application. This simple application was created using the Django web framework. So, what's that, exactly? Read on to learn more!