I'll primarily use Python since it's a powerful, yet easy-to-learn language that's used in many applications. There are also numerous well-supported libraries for Python that can be used to help you quickly accomplish your task. We'll learn about some of those libraries in this course and how to use them

use version 3.6 of Python, and so it's recommended that you install this version if you want to follow along. You can find the install for this on the python.org website. There are also a few other tools that I'll be using in this course.

a console emulator Cmder, as my command prompt.

http://cmder.net.

any text file editor Visual Studio Code.

code.visualstudio.com

Chrome developer tools

code inspector

**REST API calls from Python**

The idea of an API can be intimidating but the reality is getting started with it in Python is actually quite easy. But before we do that let's take a moment to quickly overview what it means to have a REST API. Basically it's just a way to interact with a site by telling it what to do.

You specify a URL to indicate the things you want to use and then you can ask it to do one of four things.

POST to create the object If the object does not exist

PUT to modify the object if the object already exists

DELETE which would remove the existing object If the object already exists

GET some information about an object and if the object exists

But let's see this in action, well switch over to the command console and let's install a module that will help us work with REST APIs in Python. We can do this with pip in Python very easily. We'll just call pip install and we'll install this module which is the request module.

python -m pip install requests

This'll run for a second and download the dependencies that we need and we're ready to go. So to work with this module we'll use this site called JSONPlaceholder.

<http://jsonplaceholder.typicode.com>

It's a site that let's us quickly and easily try out some actions for REST APIs. So let's try this out. Let's start Python and let's import the request module that we just installed.

***python***

***import requests***

Now for our first stab at it we want to just get some information about one of the objects that's already created on this site.

So if we go back to the site and we take a look at it. It provides us with some different resources that have already been populated. So we'll take a look at the photos and if we click on this this shows us what the API will return.

http://jsonplaceholder.typicode.com/photos

And let's see if we can set that up and access that information through Python. So we'll switch back to the command console and we'll define our URL which is the first thing that we need. So our URL will just be the URL of the site here

**url = 'http://jsonplaceholder.typicode.com/photos'**

and then we want information about the photos.

So we'll add photos to that and that will give us our URL. Now let's define a variable called response that we can read our data into and we'll use the request module to do a GET command on it and we'll get the information about that URL.

***response = requests.get(url)***

Now we can print this out by looking at the JSON of this response and let's see what we get.

***print (response.json())***

That prints out a whole bunch of information about the photos and there we have successfully made our first API call.

**do a POST or a PUT.**

need to define a JSON payload that will define the information about that object.

a JSON payload - a Python dictionary.

album ID:1

title ‘test’

URL: nothing.com because this is just a test sample

thumbnailURL: nothing.com.

***jsonPayload = {'albumId': 1, 'title':'test', 'url':'nothing.com', ‘thumbnailUrl':'nothing.com'}***

use that in a POST request.

Save the response to that to a variable equals request dot post and we'll post to the same URL but this time we will give it a JSON argument that takes in our JSON payload

***response = requests.post(url,json=jsonPayload)***

if we look at

***response.json()***

We can see that we've added a new photo with the ID of 5001 so now there's an additional photo in the album.

***{'albumId': 1, 'title': 'test', 'url': 'nothing.com', 'thumbnailUrl': 'nothing.com', 'id': 5001}***

PUT is very similar to POST but need to PUT the information to a particular object; need to modify our URL; modify photo 100.

***url = 'http://jsonplaceholder.typicode.com/photos/100'***

Change our URL, we'll use the same JSON Placeholder info that we did last time. We will once again assign this to our response variable and we'll call the request module with PUT.

We use our URL and we'll give it the JSON argument which is our JSON payload

***response = requests.put(url,json=jsonPayload)***

and then let's call

***response.json()***

to see what has happened.

We can see here that we've modified photo ID 100 to include the new information that we defined.

{'albumId': 1, 'title': 'test', 'url': 'nothing.com', 'thumbnailUrl': 'nothing.com', 'id': 100}

**the delete command.**

assign this to the response variable.

***response = requests.delete(url)***

now if we look at the

***response.json()***

we can see that it's blank

*{}*

because that item no longer exists and there you have it.

You now know how to work with the four main API calls.

**Authentication**

We looked at how easy it was to get set up using REST APIs in Python. However, when we did that, we did it on a site where we didn't need to log in. Often though, we're dealing with data that's protected by authentication. In this case, we need to do a little bit more work, but the Python request module still makes this pretty easy for us. In the next example, I'm going to use a GitHub account I created for testing purposes. Let's first try to get some information about the user doing the same thing that we did the last time. So let's define a URL. In this case, it will be https api.github.com/user.

And now, let's create a response variable and use the request module to get information about that URL. Let's see what this response gave us back.

***import requests***

***url = 'https://api.github.com/user'***

***response = requests.get(url)***

***response.json()***

And we see a message here that says it requires authentication.

{'message': 'Requires authentication', 'documentation\_url': 'https://developer.github.com/v3/users/#get-the-authenticated-user'}

So this is telling us that we need to give it some credentials. So let's do that. All I need to do here is to add, just go back to this request, and we just need to add authentication to it.

So we'll do auth and we'll give it our username, which in this case is djw-test. We'll give it a password. I've created this highly secure password called password1.

***response = requests.get(url,auth=(‘djw-test','password1'))***

***response.json()***

And now, let's see what we got back from our response this time. And so we can see that now, we have access to that info. So it's pretty straightforward, and if you're just testing something on your machine across a trusted network, this is fine.

But as you can see, I had to enter my **password in plain text**, which is **Not a good idea if this is something anyone else might see.**

So many sites will use **authorization tokens.**

These can be **set up to limit the access**

and **reduce the risk of damage to your site.**

So if we return to the GitHub example we just looked at, we can go to the settings in GitHub,

https://github.com/settings/profile

go to developer settings

get a Personal access token.

click generate a new token.

We'll limit the scope of this token to users

click **[ ] user**

readuser

user:email

user:follow

, since we're interested in user information,

Token description.

user token

click [Generate token]

Generate the token. And let's copy that token and return to Python. And now, we can modify our previous command, and rather than using auth, we will use headers, and headers will take in an dictionary, which will have the ***authorization key***, and then for the key, we'll prepend this with Bearer, so that we know what kind of key it is, and we'll paste in the key that we copied from GitHub.

response = requests.get(url,headers={'Authorization':'Bearer ***authorization key*** '})

response.json()

So now, let's go ahead and see what came back from that request. And we can see that we got the same information as we had when we used our username and password.

So this shows you that there's other ways to limit the access that people have to information. There are even more complicated ways to do authentication, but these two methods should be enough to get you started in most cases.

As with all things security-related, please be careful that you're **not exposing your password to an insecure connection**. But you should now be able to access data in your application, even if it is password-protected.

**Analyzing the data: Parsing data**

an overview of some of the basic things that are available to us in analyzing data. I've already set up a request here that gets a user's information from the GitHub API, so let's take a look at that request.

We'll print out the response.json for this and we can see that it's wrapped in these curly braces.

These curly braces indicate a Python dictionary.

A dictionary is one of the basic data structures in Python and there are a few helpful things to know when using them. They're made up of key value pairs that look like this,

key:value

where the colon in between represents a mapping from the key to its value.

These can be almost arbitrary values and can also be nested with multiple dictionaries inside of each other. And so as you can imagine, they can contain a lot of information.

But instead of using the response.json going forward let's assign that to a variable. So we'll assign it to this variable called my\_json which will be a Python dictionary. So we'll give response.json to that variable. And now let's take a look at this dictionary, and let's print out all the keys.

***my\_json = response.json()***

So to do that we can loop over the keys of the dictionary, for key in my\_json.keys, and then we will just print each key.

***for key in my\_json.keys():***

***print(key)***

hit Enter twice

it will run through all the keys in the dictionary.

And we can see that got a long list of the different keys that are available in the dictionary. Now if we want to find the value that corresponds to a particular key, we can just call the dictionary with that value.

So we can do that like this: my\_json, and then we'll give it the key, so we'll use id here,

**my\_json['id']**

and we can see that it prints out the number that represents our ID.

There is of course, a lot more that you can do with data in Python, but to be honest, the best way for you to figure it out is to try and solve a simple problem on your own. You have enough information now to know how to do that, so at this time we can move on to an exercise that will help us cement what we've learned so far.

Challenge

will use data from the JSONPlaceholder site that we've looked at before, and your challenge will be to see if you can

**find any photos that use duplicate URLs.**

So if we take a look at this data set, we can see that each photo has a URL associated with it, and in this challenge, we want to go through and see if there are two or more photos that are referencing the same URL.

Don't worry too much about what particular photos are pointing to the same URLs. All we want to know is if there are any instances of this happening in this data set. One hint in doing this in Python is that if you have a list of items that has duplicates in it, and you turn that list into a set, the set operation will remove all the duplicates from the list. So

let's start Python

python

create a list

list1 = [1,2,3,1,2,3]

turn this list into a set just by calling set on it.

set(list1)

We can see that the duplicate entries have been removed from it.

{1,2,3}

And so if we were to look at the length of the list and the length of the set, we would be able to see if there were any duplicates in the list. There's a file called Challenge1 in the exercise files that has some of the basics of this filled in to get you started.

Challenge1.py

When I was showing you examples earlier, I was showing them using the interactive command prompt. And this is a powerful feature that Python has that allows you to easily try things out and experiment.

more complex scripts, save them in a file and run them from there.

calling cd to change directories into the directory

*cd 01\_04*

run file type

*python Challenge1.py*

# Challenger Solution

# Problem description

# Find out if there are any duplicate urls used in the

# json placeholder photo data

import requests

url = 'https://jsonplaceholder.typicode.com/photos'

#get the data about the photos

response = requests.get(url)

#read that data into a variable

json\_data = response.json()

#create a list for storing the url of each photo

url\_list = []

for photo in json\_data:

url\_list.append(photo['url'])

#How many items are in the url list (Should be 5000 since we have 5000 photos in our dataset)?

print(len(url\_list))

#How many items are there if we turn that list into a set?

print(len(set(url\_list)))

Alternative tools for API scripting

some other tools that are available to help you out with API testing.

1 Postman. https://www.getpostman.com

One very powerful and easy to use tool

a powerful Chrome browser plug in that allows you to make some requests and to see the responses in the UI. It's a very helpful tool to use when you're doing some initial exploration of your API.

using a GET request by default. If we click on GET \/ dropdown

a lot of different options available.

GET, POST, PUT, DELETE

But those four still do cover most of the situations. But if you wanted to dive in deeper into your API, there are a lot of different options available here as well. Postman could be a great place to help you figure out and understand some of those options, if they are available in your API.

But for now, we'll just keep using the get request and let's do something similar to what we did when we were using this in Python. We'll use this URL to our Json placeholder site. So, you see I've typed this before, it's auto finished it for me.

GET http://jsonplaceholder.typicode.com/photos

Just click on this and we'll send that request and see what happens. And so we can see that we got back here in the body, a list of all the photo URLs in much the same way that we did when we ran this through Python in the command prompt.

Now the downside of this is that if we did want to do further scripting with these results, we would need to copy them out of here and into another tool. But if we're just trying to figure something out quickly, if we're just trying to take an initial look at our API, and see what the requests look like, this is a great tool to use.

As you can see, there are many different tools that are available to help you, depending on what you're trying to accomplish. There are of course many tools available to help you with using and testing APIs but I'd recommend you take a bit of time to learn how to do it with Python and perhaps poke around with Postman a bit.

And then if you run in to difficulties, like perhaps you need a tool that works in this specific language like JAVA

you look for alternative tools that can help you then.

The biggest thing in learning how to affectively use APIs is to use them, and interact with them. Once you spend some time using APIs through tools that you're familiar with, like Postman or Python, you'll find that it's much easier to pick up new tools.

**Automate Report Generation Using Google APIs**

Gathering data

Testers need to find out data about applications, but they also often need to report that data to others, so let's take a look at some of the

tools that we can use to help us automate report generation.

We'll be looking at some test timing data for test runs.

In the exercise files, TestTimingData.csv CSV file

that has a bunch of data made up about a fictional set of test runs.

how we can create a column chart showing how much slower or faster a current run is from the average runtimes for that test

how we can store the historical data in a table so that we can look back through it later if we want to.

# import csv module

import csv

#final desired format

# - Charts [["Test Name",<diff from avg>]]

# - spreadsheet [["Test Name",<current run time>]]

# define the list; blank

timing\_data = []

# read the file

# read in the data from our file reader,

# call for each row in file\_reader

# append it to our timing data.

# the CSV reader will split each row at the commas

# and automatically make this into a list of lists for us.

with open('TestTimingData.csv') as csv\_file:

file\_reader = csv.reader(csv\_file)

for row in file\_reader:

timing\_data.append(row)

# get the data that we want to use for each of our use cases.

# define both list of lists for each of these data sets

# so to only need to loop over the underlying list once.

# create column\_chart\_data; initialize to "Test Name", "Diff from Average" headers

column\_chart\_data = [["Test Name","Diff from Avg"]]

# create list table\_data; initialize "Test Name", "Run Time" headers

table\_data = [["Test Name","Run Time (s)"]]

# populate lists

# loop from 2nd row (i.e. row 1) to the end (i.e. ":")

# skip 1st row (i.e. row 0)

for row in timing\_data[1:]:

# 1st column

test\_name = row[0]

if not row[1] or not row[2]: # skip rows with blank data

continue

# 2nd column

current\_run\_time = float(row[1]) # convert to float

# 3rd column

avg\_run\_time = float(row[2]) # convert to float

# difference from avg runtime

diff\_from\_avg = avg\_run\_time - current\_run\_time

# for each row append list that has test\_name and diff\_from\_avg

column\_chart\_data.append([test\_name,diff\_from\_avg])

# for each row append list that has test\_name and current\_run\_time

table\_data.append([test\_name,current\_run\_time])

# print data

print (column\_chart\_data)

print (table\_data)

**Using The Google Chart API**

https://developers.google.com/chart/interactive/docs/gallery

https://developers.google.com/chart/interactive/docs/gallery/columnchart

Build a template:

https://developers.google.com/chart/interactive/docs/basic\_load\_libs

code

<script type="text/javascript" **src="https://www.gstatic.com/charts/loader.js"**></script>

<script type="text/javascript">

**google.charts.load('current', {packages: ['corechart']});**

**google.charts.setOnLoadCallback(drawChart);**

  ...

</script>