API Querying Special Topic Notes

What is an API?

API stands for application programming interface. It acts as a communication interface so different computers/systems can talk to the application hosting the API. Your operating system for your computer has APIS, Zoom will talk to them to get your camera and audio turned on, which is why when you join a new web platform you get a little pop-up message from your operating system saying the web platform is asking permission to access your camera and microphone. The web platform has made a request of your operating system API, and your operating system API wants to check with you if its ok, if it doesn't recognize the web platform from a previous encounter.

Today we are going to be focusing on web-APIS, they are designed to communicate across networks, and they are a great way to gather data from the internet.

The wide world of Web-APIs

web-APIs have been around for quite some time, emerging early on in internet history, and there are currently three main types of web API:

1. SOAP (Simple Object Access Protocol) is typically associated with the enterprise world, has a stricter contract-based usage, and is mostly designed around actions.
2. REST (Representational State Transfer) is typically used for public APIs and is ideal for fetching data from the web. It's much lighter and closer to the HTTP specification than SOAP.
3. GraphQL (Graph Query Language) is pretty new, its a very flexible query language for APIs and it allows the user to decide exactly what they want the API to fetch from the application instead of the reverse.

We are going to be working with REST APIs as they are widely available and easiest to use for beginners.

Can I use the API on *this site?*

Most sites that have public API's have terms and conditions on their use and documentation for how to access them. Some sites require you to have a special account with them before you can access their API. And to enforce that they require you to authenticate (provide them with keys or tokens when you make your requests). Most large sites, especially social media sites, including twitter, instagram etc, require you to have a special account in order to access their APIs.

But what we are going to accomplish today: learning how to query and manage data from a less complex site and API, then walking you through a notebook that is designed to query from the new Twitter API. I will also walk you through the twitter documentation. developer portal, and how you would go about getting a special access account with them.

Getting setup

We are going to use google collab today. You need a gmail account to open the workbook, it can be a throwaway account if you wish, or if you have another environment you want to use let us know and we can share the code file with you another way. Click on the link in the chat to open google collab:
Click: copy to Drive
Now lets import the libraries you need.

```
import requests  #need this to interact with the API
import json  #need this to help with formatting the response
from pprint import pprint #will make things easier for us to read
import time  #will allow us to introduce time delays in our program
```

In this special topics workshop the first site we are going to query a site called "excuser" https://excuser.herokuapp.com/

This website has an open API and all the necessary documentation is on the landing page. The website provides you with excuses for different occasions.

When you call(or query) an API your are making a "request". When you make a request of an API it will always send you a response.
Requests contain relevant data regarding your API request call, such as the base URL, the endpoint, the method used, the headers, and so on.
Responses contain relevant data returned by the server, including the data or content, the status code, and the headers.

Lets do a simple request and see what we get. We are just going to do a request with the base URL:

In a new cell type:

```
base_url = "https://excuser.herokuapp.com/v1" #this is the base url of the excusersite
requests.get(base_url) #try a request just from the base URL
```

From this we should get a response, and because we haven't asked for any data it is just going to give us a response code: 200, means OK.
https://www.restapitutorial.com/httpstatuscodes.html here is a site where you can look up what the different codes mean, most of us have seen the 404 error when we type in a website into our search engine that doesn't exist.
So this is good, that means the API is working, we have the correct base address and it does not require authentication.

Endpoints

An endpoint is added to the base url and it specifies different resources- so what kind of data you are interested in. There are several endpoints listed on this site under "usage"
Let's try the first one /excuse, we should get a random excuse

If we just send the request like we just did before we are still just going to get a response code. But we want to save the data:

```
random_excuse = requests.get(base_url+"/excuse") #try a request from the "excuse" endpoint
```

Now we have saved this request as a request object to the variable "random_excuse". So we can use some methods from the requests library to explore different aspects of it. In a new cell type:

```
random_excuse.status_code
```

Now try:

```
```

```
random_excuse.headers
```

Now lets try

```
random_excuse.text
```

The output for the .text function is actually JSON (Java script object notation). And if we apply a JSON method we can export this as a dictionary, which will be much easier for us to work with:

```
random_excuse.json()
```

lets try specifying how many excuses we want:

```
many_excuses = requests.get(base_url+"/excuse/10") #try a request from the
"excuse" endpoint
```

Querying

What if we want to do more than just rely on the defaults? How do we add query parameters? In a new cell?

Lets try something a bit more complicated : https://metmuseum.github.io/
This API doesn't require authentication but it has a lot of different end points and parameters. We are going to look more closely first at the 'search' endpoint. It says is "returns a listing of all Object IDs for objects that contain the search query within the object's data". So we can specify some criteria on the museum objects we are interested in, and it will return a list of object IDS for the objects that match our criteria. In a new cell type:

```
base_url_met = "https://collectionapi.metmuseum.org/public/collection/v1"
ocean_obj = requests.get(base_url_met+"/search?q=ocean&medium=Sculpture")
#try a request from the /search endpoint with several conditions in
dictionary form
ocean_obj.json()   #show the result as a dictionary
```

If we have a lot of parameters that we want to use over and over we can write this slightly differently:

```
search_cond = {'q':'ocean','medium':'Sculpture'}
base_url_met = "https://collectionapi.metmuseum.org/public/collection/v1"
ocean_obj = requests.get(base_url_met+"/search", params=search_cond) #try a request from the /search endpoint with several conditions in dictionary form
pprint(ocean_obj.json())   #pretty print the result as a dictionary
print(ocean_obj.url)      #print the url that is being used in the query
```

This format may not be the most helpful to have the information in, we have just a list of the object ids, and don't know the titles or descriptions for those objects and we can't get that information from this API endpoint. We need to extract that list of ID's and feed that back into the objects endpoint to get more information.

Formatting and extracting data

In a new cell type:

```
list_ids =ocean_obj.json()['objectIDs']
print(list_ids)
```

That list looks good, now in a new cell lets build a 'for loop'

```
ocean_rep = {}
for item in list_ids:
   resp = requests.get(base_url_met+"/objects/"+str(item))
   d = resp.json()
   ocean_rep.update({(item,d['title']):d})
   print((item, d['title']))
   time.sleep(.25)
```

Let's look at the full information for one of these items

```
pprint(ocean_rep[(206836, 'The Flood')])
```

Authentication

What about API's that require authentication? A lot of API's require you to ask permission to use them and then they provide you with some type of authentication method when you make requests. Twitter, instagram, facebook etc lots of big sites, especially on social media require you to authenticate to query their API. Some sites have longer, more rigorous processes than others, so it's good to plan for that.

Different sites have different methods for dealing with authentication, usually all the information you need is contained in the API documentation.
If you want to try some more open, free API's, below is a link to a great list. Not all API's are created equal, and some aren't maintained so, something to keep in mind.
https://github.com/public-apis/public-apis

Lets talk about twitter

To query twitter you need a developer or researcher account. If you are doing research with twitter data, apply for the academic researcher account, it has a much much bigger limit as to how much data you can pull down at once, and has extra features.

Academic research account information:
https://developer.twitter.com/en/products/twitter-api/academic-research

Regular developer account:
https://developer.twitter.com/en/docs/twitter-api/getting-started/getting-access-to-the-twitter-api

Once you have made it through the application process (took a few months for me) then you get access to the developer portal:
https://developer.twitter.com/en/portal/dashboard

Good news! organizationally, the documentation on the new twitter API has improved significantly over the last year. So uif you, like me, were starting to dabble with it over the last year and were getting really frustrated with the API documentation, good news, it has gotten better!

If you open the drop down for each of these features, it will tell you what it does, what the endpoints are, how often you can query them, whether there is a cap to how many tweets you can pull, and a link to more detailed documentation.

We are gonna use search tweets.

Note here as we are reading through the documentation that features like a full-archive search are only available for academic research access.

If you want to fiddle around with different query parameters etc, and you aren't really comfortable in python, and you want to do a lot of quick requests: get postman

I'll show you what that looks like, the quick start guide does a great job of walking you through that.

If you want to pull tweets for a project, you want to code it, and preferably not reinvent the wheel:
https://github.com/twitterdev/Twitter-API-v2-sample-code

click on the green "code" button and click "download ZIP". extract that folder to your computer, somewhere you can find it, like your documents, then navigate to that folder in your jupyter lab

It contains sample code, for all the basics, you will have to copy and paste it into a jupyter notebook if you want to use it that way.

I have gone through it and rewritten some of it, adding comments etc. to make it a bit more usable. This code book I have put together, does a recent tweet search, based on criteria you can adjust, then saves the results to a csv file.

github link:
https://github.com/lvermeyden/programming_lab_code/blob/cd7550fbc8af49f3cba6a31f9ab7517f1310f268/TwitterAPI2_Feb_2022.ipynb

collab link:
https://colab.research.google.com/github/lvermeyden/programming_lab_code/blob/main/TwitterAPI2_Feb_2022.ipynb