24.9. Debugging calls to requests.get()

24.9.1. In the Runestone environment

In our limited implementation of the `requests` library for the Runestone environment, if your request fails for any reason, you will still get a Response object. Most likely, you will realize there was a problem because you invoke the `.json()` method on the Response object and you get an error that refers to an "unexpected token" in the JSON. When that happens, you should print out the values of the `.text` and `.url` attributes.

1. If it was unable to use your baseurl and params value to create a url string, the `.url` attribute will be "Couldn't generate a valid URL" and the `.text` attribute will be set to "<html><body><h1>invalid request</h1></body></html>".
    - In that case, you should check that the value you passed for the `params` parameter is indeed a dictionary and that its keys and values are all strings.
2. If it generated a url string but failed to fetch data from the server described by the url string, the `.url` attribute will be set correctly and the `.text` attribute will be set to "Failed to retrieve that URL".
    - In that case, you should examine the url to try to figure out what went wrong. In particular, print it, then copy and paste it to a browser window and see what happens.

24.9.2. In a full python environment

In a full python environment, you will not always get a Response object back from a call to `requests.get`. What you get back will generally be even more informative than what you get in the Runestone environment, but you have to know where to look.

The first thing that might go wrong is that you get a runtime error when you call `requests.get(dest_url)`. There are two possibilities for what's gone wrong in that case.

One possibility is that the value provided for the `params` parameter is not a valid dictionary or doesn't have key-value pairs that can be converted into text strings suitable for putting into a URL. For example, if you execute `requests.get("http://github.com", params = [0,1])`, [0,1] is a list rather than a dictionary and the python interpreter generates the error, `TypeError: 'int' object is not iterable`.

The second possibility is that the variable `dest_url` is either not bound to a string, or is bound to a string that isn't a valid URL. For example, it might be bound to the string `"http://foo.bar/bat"`. foo.bar is not a valid domain name that can be resolved to an ip address, so there's no server to contact. That will yield an error of type requests.exceptions.ConnectionError. Here's a complete error message:

```
requests.exceptions.ConnectionError: HTTPConnectionPool(host='foo.bar', port=80): Max retr
ies exceeded with url: /bat?key=val (Caused by <class 'socket.gaierror'>: [Errno 11004] ge
taddrinfo failed)
```

The best approach is to look at the URL that is produced, eyeball it, and plug it into a browser to see what happens. Unfortunately, if the call to `requests.get` produces an error, you won't get a Response object, so you'll need some other way to see what URL was produced. The function defined below takes the same parameters as `requests.get` and returns the URL as a string, without trying to fetch it.

```
import requests
def requestURL(baseurl, params = {}):
    # This function accepts a URL path and a params diction as inputs.
    # It calls requests.get() with those inputs,
    # and returns the full URL of the data you want to get.
    req = requests.Request(method = 'GET', url = baseurl, params = params)
    prepped = req.prepare()
    return prepped.url

print(requestURL(some_base_url, some_params_dictionary))
```

Assuming `requestURL()` returns a URL, match up what you see from the printout of the params dictionary to what you see in the URL that was printed out. If you have a sample of a URL from the API documentation, see if the structure of your URL matches what's there. Perhaps you have misspelled one of the API parameter names or you misspelled the base url.

You can also try cutting and pasting the printed URL into a browser window, to see what error message you get from the website. You can then try changing the URL in the browser and reloading. When you finally get a url that works, you will need to translate the changes you made in the url back into changes to make to your baseurl or params dictionary.

If requests.get() executes without generating a runtime error, you are still not done with your error checking. No error means that your computer managed to connect to some web server and get some kind of response, but it doesn't mean that it got the data you were hoping to get.

Fortunately, the response object returned by requests.get() has the `.url` attribute, which will help you with debugging. It's a good practice during program development to have your program print it out. This is easier than calling `requestURL()` but is only available to you if `requests.get()` succeeds in returning a Response object.

More importantly, you'll want to print out the contents. Sometimes the text that's retrieved is an error message that you can read, such as `{"request empty": "There is no data that corresponds to your search."}`. In other cases, it's just obviously the wrong data. Print out the first couple hundred characters of the response text to see if it makes sense.

```
import requests
dest_url = <some expr>
d = <some dictionary>
resp = requests.get(dest_url, params = d)
print(resp.url)
print(resp.text[:200])
```

Now you try it. Use `requests.get()` and/or `requestURL()` to generate the following url, `https://www.google.com/search?tbm=isch&q=%22violins+and+guitars%22`. (Don't look at the previous page of the textbook, at least not yet. If you can't figure it out after 15 minutes of trying the approaches on this page, then look back.)

Check your understanding

requests-8-1: If the results you are getting back from a call to `requests.get()` are not what you expected, what's the first thing you should do?

- ○ A. look at the .url attribute of the Response object

B. look at the first few characters of the .text attribute of the Response object

C. look at the .status attribute of the response object

D. look carefully at your code and compare it to the sample code here

Check me Compare me

✔ Checking the contents that were returned is a good first step

Activity: 1 -- Multiple Choice (question400_5_1)

requests-8-2: In a full python environment, if there is a runtime error and you don't get a Response object back from the call to `requests.get()`, what should you do?

A. look at the .url attribute of the Response object

B. look at the values you passed in to requests.get()

C. invoke the requestURL() function with the same parameters you used to invoke requests.get()

D. look carefully at your code and compare it to the sample code on this page

Check me Compare me

✔ It can be really helpful to see the URL that was generated; you may be able to spot what went wrong

Activity: 2 -- Multiple Choice (question400_5_2)

requests-8-3: In the runestone environment, if there is a runtime error and you don't get a Response object back from the call to `requests.get()`, what should you do?

A. look at the .url attribute of the Response object

B. look at the values you passed in to requests.get()

C. invoke the requestURL() function with the same parameters you used to invoke requests.get()

D. look carefully at your code and compare it to the sample code on this page

Check me Compare me

✔ Generally, a runtime error when you invoke ``requests.get`` in the Runestone environment is caused by the value of the ``params`` parameter not being a dictionary, or not having only strings as keys and values.

Activity: 3 -- Multiple Choice (question400_5_3)

You have attempted 4 of 3 activities on this page

24.8. Figuring Out How to Use a REST API">

guring Out How to Use a REST API">

24.10. Requests Cookbook">

✔ Completed. Well Done!

24.10. Requests Cookbook">Next Section - 24.10. Requests Cookbook

| Back to top