Lab 2 Pinterest API, JSON, and HTML

Brief

- Due date: 11:59PM 2-20-16 (the next Saturay after the lab)
- Stencil: /course/cs1951a/pub/labs/api/stencil
- Handin: cs1951a_handin api
- Required files: README, code/, html/

Overview

In this lab we will demonstrate how to use the Pinterest API to download pin information. We will also go over using JSON, which is the format in which your pin information will be returned. You will then combine this information to determine whether your lab TAs are more excited about lamps or IKEA meatballs.

The pins you will be looking at come from our Pinterest account (https://www.pinterest.com/bdatascience/), which we've used to pin all the Ikea-related things that we've liked.

Setup

- Get your Pinterest Developer information
  - First, you'll need access to the Pinterest API. You'll have to have a Pinterest account in order to do this. You can create one here (https://www.pinterest.com/join/).
  - Once you have your account, visit this link (https://developers.pinterest.com/tools/access_token/) to get an access token, using the default scopes. An access token gives you permission to use the API. This access token will be valid forever, so hold on to it.
  - If you're developing locally, and you haven't installed anaconda like we've recommended, you'll need to install the requests package. To do so, type `pip install -U requests` into the command line.
- You will need to modify
  - `pinterest.py`
  - `pins.py`
  - `meatballs_lamps.py`

Data

The `/course/cs1951a/pub/labs/api/stencil/stencil.tar` zip file contains all the data and stencil code you will need. First, you'll want to copy this stencil into your course directory.

If you haven't created a directory for this course, we recommend that you do so immediately. You can do so by typing:

```
cd ~/course
mkdir cs1951a
cd cs1951a
mkdir labs
cd labs
```

`cd` means to change directory. `mkdir` makes the named directory in the current directory. You can see the contents of the current directory with `ls`.

To copy the stencil into your directory, type

```
mkdir api
cd api
cp /course/cs1951a/pub/labs/api/stencil/stencil.tar .
```

`cp` copies the first argument (a file), creating a new file with the name of the second argument. `.` means to create a copy of the file with the same name.

To unzip the file, type `unzip stencil.zip`. Now you have all the files you will need for this lab!

Assignment

Introduction: Anatomy of a Pin

Pins are JSON objects. JSON (JavaScript Object Notation) is a lightweight data-interchange format. It's easy for machines to parse and is completely language independent (don't let the name fool you). JSON is built on two kinds of structures: objects, which are collections of name-value pairs, and arrays, which are ordered lists of values.

JSON Formatting

JSON has a few kinds of values:

- Strings: a series of characters surrounded by double quotes.
- Numbers: a floating point number like 4, 3.14, 10e5, with nothing surrounding it.
- True
- False
- Null
- Objects
- Arrays

Objects in JSON begin with the '{' and continue until a matching '}' is found. Within the object is a series of name-value pairs separated by commas. Names are followed by a ':' and then their value. Like this:
`{"name": "value", "another name": "another value"}`

Arrays in JSON are ordered collections of values. They begin with '[' and end with ']'. Values are separated by commas. Values within an array do not need to be all of the same type. Like this:
`[1, "string", true, null, {"look": "an object"}]`

You'll need a basic understanding of JSON for this lab and the rest of the course. Python kindly provides us with a JSON module (https://docs.python.org/2/library/json.html). You'll get familiar with this module as we use it, but if you get stuck (or have extra time during the lab) we recommend looking at it.

These are some important JSON commands:

- Import JSON
  ```
  import json
  ```
- Parse String into JSON:
  ```
  json.loads('["foo", {"bar":["baz", null, 1.0, 2]}]')
         [u'foo', {u'bar': [u'baz', None, 1.0, 2]}]
  ```
- Access a JSON data field:
  ```
   data = json.loads('{"lat":444, "lon":555}')
   print(data["lat"])
         444
  ```
- Write JSON to String:
  ```
   print json.dumps("\"foo\bar")
         "\"foo\bar"
  ```

Pin Data Fields

Pins have lots of fields (though many of them won't be relevant). These are the fields of an example pin about Swedish Meatballs:

- attribution: null
- creator:{} 4 items
  - url: https://www.pinterest.com/bdatascience/
  - first_name: Data
  - last_name: Science
  - id: 517632688329465302
- url: https://www.pinterest.com/pin/517632550901647712/
- media: {} 1 item
  - type: image
- created_at: 2016-01-23T19:31:06
- original_link: http://www.paleofondue.com/2014/10/24/paleo-swedish-meatballs/
- note: Paleo Swedish Meatballs just like Ikeas. Only better
- color: #c19372
- link: https://www.pinterest.com/r/pin/517632550901647712/4779055074072594921/eb897e5298d9a3105fc50b97df4f96fa5577ce6bbe7750de8fb529
- board:{} 3 items

  - image:{} 1 item
  - original:{} 3 items
  - counts:{} 3 items
- likes: 0
- comments: 0
- repins: 0
- id: 517632550901647712
- metadata:{} 0 items

As you can see, this contains some useful information, like the name of the creator, the comments written about it, and the original link to the image used in the pin. Some fields have subfields, like the creator field. We can comfortably ignore most of this stuff. In general, you will always want to decide what information you need, and then you can look at the data you have to figure out how to access it.

Part 1: Downloading Pins

How to use a web API

First, we'll begin with a quick explanation of how to use a web API. APIs give you access to a web service through structured calls. Rather than download an entire page from Pinterest, we can instead use their API to download Pins in a structured format.

To use Pinterest's API, well need to make requests to special URLs. HTTP defines a set of verbs corresponding with different kinds of requests. You can see a full list here (https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods). We're mostly interested in the GET request, which gives us a representation of a specific resource.

To perform requests in Python, we'll be using the requests module (http://docs.python-requests.org/en/latest/). As mentioned in setup, you'll have to install this on your own machine but it is available on the Sunlab machines. This module makes performing requests very simple.

Modifications to PinterestAPI Class

We'll be creating a class called `PinterestAPI` . If you're unfamiliar with classes, read this (https://en.wikibooks.org/wiki/A_Beginner%27s_Python_Tutorial/Classes#Creating_a_Class), specifically "Creating a Class", "Using a Class", and "Lingo".

Open up the stencil file `pinterest.py` and we can begin. For now, you should have a mostly empty class. We've provided you with a class declaration and the `__init__` method. When we create a `PinterestAPI` object, we need to pass in an access token. The class will then use that access token for all requests it makes.

We're interested in the portion of the Pinterest API that can retrieve all pins for a given board. Take a look here (https://developers.pinterest.com/docs/api/boards/#fetch-board-data). Specifically, we'll want to make a GET request to https://api.pinterest.com/v1/boards//pins. Note that we want to make the request which will give us the pins.

First, we'll want to fill out the `get_request` function. This will allow us to pass in a URL and a dictionary of parameters and make a request based on those. Let's look at how requests performs requests:
`r = requests.get(«pinterest-URL», params=params)`

All we need to do is pass in the URL and parameters. This then returns a Response object, which allows us to work with the data the server sends us. We'll be interested mostly in the `json()` method of the response object, which converts the response into a JSON object.

Notice that in `get_request` we add the access token to our dictionary of parameters:
`params['access_token'] = self.access_token`

This ensures that any call we make will have the correct token. Next, we append our prefix URL ('https://api.pinterest.com/v1/') with the rest of the API URL. `url = self.prefix_url + path`

Now you need to perform a get request on the constructed URL (`url`) using the specified parameters (`params`). Then return the JSON object form of the response.

Get request parameters

Now that we have a generic get request function, making a request on a specific part of the API is easy. First, let's see what we need to give as parameters to get all of a board's Pins. The Pinterest API tells us that we need to provide only an access code and a list of fields we want to retrieve. Our function defaults to `fields=None` specifying that we want to use the default fields during our request.

Take a look at function `get_board_pins` . We've filled in most of this function for you. Notice how it depends on `get_requests` to make the actual request. All we need for you to do is provide the proper path to access the API. We want to access https://api.pinterest.com/v1/boards/«board»/pins. Remember that in `get_request` , https://api.pinterest.com/v1/ is added to whatever you pass in.

Finally, let's make a request! Go look at the `main` function in the script. This is will be run whenever you type `python pinterest.py` . Let's first create a PinterestAPI object using your access token. Next, make a request to the board `bdatascience/ikea-lab` using `get_board_pins` .

Once you've done that, take a look at our board (https://www.pinterest.com/bdatascience/ikea-lab/) and enjoy all of the pretty lamps and yummy meatballs.

Getting all Pins

Now, let's go back to the data you just pulled. Get the data from the request, indexing the JSON object on `data` . This will be an array of pins. Now, print the length of that array. Notice that its length is only 25. That's not right, is it? Let's fix that.

By default, the Pinterest API only returns 25 requests. You can provide a `limit` parameter to expand that to up to 100, but we want more than that. However, each request also yields a `page` field, which has a `next` field containing a link to the next request (this link is the url and params rolled into one; you can just call `request.get` on the link). When we run out of pages, `json_object['page']['next']` becomes null. Using this information, modify `get_all_board_pins` to get all of the pins from a board. We've already written the part that performs the initial request and adds that information to an array called `data` . Modify the loop, which runs while there's still another page, to make a new get request using `requests.get` and add that data to `data` . Print the length of the data array to make sure it's length is 33.

We now want to save this json data; save it to a file called `pins.json` . Take a look at the next section of this lab for how to do this.

Part 2: Converting from JSON and Writing to CSV

A CSV file is a text file with a very specific format: each row other than the first row is an entry, and for each entry, fields are separated by commas. The first row contains comma-separated field headers. Below are example contents for a CSV file:

```
  Name, Age, Class Year
  Sarah, 18, 2019
  James, 20, 2017
  ...
```

The first step to reading or writing to a file is opening it. You can do this with the open command: `f = open('filepath', 'w')` . The first element is the name of the file and the path to it. The second element describes the way in which the file with be used: 'w' to write, 'r' to read, 'a' to append to the existing contents.

You will be using Python's CSV reader and writer, here (https://docs.python.org/2/library/csv.html). We've provided an example of opening and writing to a CSV file, below.

To read from a CSV file:

```
import csv
with open('some.csv', 'rb') as f:
    reader = csv.reader(f)
    for row in reader:
        print row
```

To write to a CSV file:

```
import csv
with open('some.csv', 'wb') as f:
    writer = csv.writer(f)
    # Write CSV Header, If you dont need that, remove this line
    writer.writerow(["Name", "Age", "Year"])
    for entry in json_data:
        writer.writerow([entry["Name"], entry["Age"], entry["Year"]])
```

Now, you will need to parse the JSON file you obtained through the API call. The file contains lines of code where each line is a JSON object with the pin structure described earlier.

You should convert each line of your JSON file into a JSON object by calling `json.loads()` (you may also want to look at `json.load` ; yep, they're different...), and write objects that represent pins to a CSV file (follow the instructions above). You only need to store 4 attributes: *url*, *original_link*, *note*, and *id* in your output file. We provided you with a stencil, `pins.py` . Save your output file as `pins.csv` .

Note: in this lab, the JSON file is only a single line. However, many times it won't be; get used to writing your code such that it iterates over the entire file.

Final note: JSON objects use unicode encoding; you'll read more about this in the Data Integration assignment. However, csv files don't support unicode encoding (or at least don't easily). Make sure to encode your unicode strings to UTF-8 by calling `encode('utf-8')` on the unicode string so that you can write them to the csv file.

Part 3: Lamps or Meatballs?

Your task in this part is to determine how many of the pins on our Ikea Lab board refer to lamps and how many refer to meatballs. You've already downloaded all the pins on the board, parsed the JSON information, and written the data to a CSV file. Now, we will load the data from the file and search for the words "lamp" and "meatball" in the *notes* field for each pin. Look in the previous section for a reference on loading from a CSV file.

Also search for possible variations on both of these words: "Lamp" and "lamps" should both be counted. You may want to use the command `s.lower()` to convert all characters in the string "s" to lowercase. The stencil code for this assignment is in `meatballs_lamps.py` . Your script should output the number of pins referring to lamps and meatballs.

If you have the time, try to use regexps to search for the words; take a look at the python regular expression library here (https://docs.python.org/2/library/re.html).

To get points for this part (and you want points!), be sure to print out the number of lamps and number of meatballs on the same line, i.e. `print num_lamps, num_meatballs` . Be sure to print out the number of lamps before the number of meatballs.

Handing in

You only need to hand in the Python parts of this lab. The folder you hand in must contain the following:

- README - text file containing anything about the lab that you want to tell the TAs
- code/ - directory containing all your code for this lab
- code/pinterest.py
- code/pins.py
- code/meatballs_lamps.py

Then run: **cs1951a_handin api**

Part 4: HTML (Optional)

You'll be creating writeups for all future assignments using HTML. Open `html/index.html` in both a web browser and a text editor to get an introduction to HTML. It'll walk you through the very basics of HTML and how to create your writeups in the future. This section isn't graded and is entirely optional, so you're free to skip it entirely. If you have not worked with HTML or CSS in the past it is **strongly recommended** that you do this part of the lab. As it's not graded, feel free to come back to this later if you would rather leave early or are running out of lab time. If nothing else, `html/index.html` lists several good resources to get started with HTML and CSS.

Credits

Lab developed by Alexander Bertsch, Christine Whalen, and Julia Wu.