Getting-Data-I-Solutions

May 21, 2019

1 Getting Data from the Internet (Part 1)

Obtaining and processing data

Author: Dhavide Aruliah

1.0.1 Assignment Contents

- Section **??**
  - Section 1.0.4
  - Section 1.0.4
  - Section 1.0.4
  - Section 1.0.4
- Section **??**
  - Section 1.0.5
  - Section 1.0.5
  - Section 1.0.5
  - Section 1.0.5
  - Section 1.0.5
  - Section 1.0.5
  - Section 1.0.5
- Section **??**
  - Section 1.0.6
  - Section 1.0.6

EXPECTED TIME 1.5 HRS

1.0.2 Overview

Getting data from the internet requires writing code to communicate with servers to request data. It also requires being able to unpack, manipulate, and interpret the data received correctly.

In this assignment, you will focus on getting data from the web programmatically using the Python `requests` module to interact with REST (Representation State Transfer) APIs. In addition, you'll get some practice manipulating common web formats (JSON and XML) using appropriate Python libraries (`json` and `lxml` respectively).

Next week's assignment will build on these ideas to illustrate web-scraping.

The content here is drawn from Video lectures 7-1 through 7-6.

1.0.3 Activities in this Assignment

- Using `requests` to query REST APIs
- Using `json` to unpack JSON data into regular Python objects
- Using `lxml` to unpack XML data into regular Python objects

1.0.4 Using `requests`

As discussed in the video lectures, the Python module `requests` provides utilities for querying remote resources on the internet. You've seen how to use the function `requests.get` to obtain a `Response` object. The response object has numerous useful attributes that inform you of the results of the query.(including `status_code`, `encoding`, and others). You can find out more from the Python `requests` documentation.

Section 1.0.1

Question 1 For practice, you'll query a few APIs provided by `api.open-notify.org` that monitor the motion of *International Space Station* (ISS) as it orbits the earth. The `iss-pass` API provides the times at which the ISS will pass specified latitude-longitude coordinates. The URL is provided below as `URL_1`. Your task is to import the Python `requests` module, apply the `requests.get` function to obtain a `Response` object called `response_1`, and to query its status code. You should obtain a status code that indicates a problem because the URL provided does not include the required geographic latitude-longitude parameters.

```
In [ ]: ### GRADED
        ### Import the requests module with a suitable 'import' statement.
        ### Use the get function from the Python requests module to query
        ###    the URL http://api.open-notify.org/iss-pass (prepared for you as URL_1).
        ### Assign the object retrieved to response_1
        ### Assign the status code of response_1 to status_1
        URL_1 = 'http://api.open-notify.org/iss-pass'
        import requests
        response_1 = requests.get(URL_1)
        status_1 = response_1.status_code

        ### For verifying answer:
        print('status_1: {}'.format(status_1))
```

Section 1.0.1

Question 2 As in Question 1, your task here is to query a URL provided (URL_2 this time), to bind the object returned to an identifier (response_2 this time) and to extract its status code (status_2 this time). This is not a REST API, but a generic HTML page for the last page of the Internet. As this URL is correctly specified and the appropriate resource can be found, the status code returned should be 200.

```
In [ ]: ### GRADED
        ### Use the get function from the Python requests module to query
        ###    the URL http://www.1112.net/lastpage.html (prepared for you as URL_2).
        ### Assign the object retrieved to response_2
        ### Assign the status code of response_2 to status_2
        URL_2 = 'http://www.1112.net/lastpage.html'
        response_2 = requests.get(URL_2)
        status_2 = response_2.status_code

        ### For verifying answer:
        print('status_2: {}'.format(status_2))
```

Section 1.0.1

Question 3 Following from Question 2, you'll extract the text encoding from response_2 as encoding_2. Then, you'll decode the content of response_2 (using the default utf-8 encoding) and bind the result to the identifier text_2.

```
In [ ]: ### GRADED
        ### Extract the text content from the response object response_2:
        ###    Assign the encoding to the identifier encoding_2
        ###    Assign the decoded content of response_2 to the identifier text_2
        ### (The results of both should be Python strings).
        encoding_2 = response_2.encoding
        text_2 = response_2.content.decode()

        ### For verifying answer:
        print('Encoding of response_2: {}'.format(encoding_2))
        print('Content of response_2:\n\n{}'.format(text_2))
```

Section 1.0.1

Question 4 The preceding question yielded a text string formatted as HTML. Data shared on the internet is often not in HTML; it could be formatted, for instance, using JSON (JavaScript Object Notation or XML (eXtensible Markup Language formats.

In this question, you'll query api.open-notify.org/astros.json (provided as URL_astros) to obtain text content formatted as JSON. For the moment, rather than parsing it directly into Python data structures, you'll save the data to a file astros.json in the current working directory. The data provided tells you the number of astronauts currently in space along with their names.

```
In [ ]: ### GRADED
        ### Retrieve a file recording the number of people currently in space
        ###   (on the International Space Station)
        ###   Assign the response object to response_astros
        ###   Assign the status to status_astros
        ###   Extract and decode the data retrieved, & write into a file 'astros.json' in the
        URL_astros = 'http://api.open-notify.org/astros.json'

        response_astros = requests.get(URL_astros)
        status_astros = response_astros.status_code
        with open('astros.json', 'w') as astros_file:
            astros_file.write(response_astros.content.decode())

        ### For verifying answer:
        print('status_astros: {}'.format(status_astros))
        %ls -l astros.json
```

Section 1.0.1

1.0.5 Using json

The json module provides useful tools for working with JSON data. In particular, the function json.loads parses a string containing a valid JSON object and returns a Python data structure. This is particularly convenient because the string is parsed for you recursively. The function json.dumps foes the opposite; that is, it parses a nested Python data structure (typically a dict or a list) and produces a valid string with a JSON description of the object (nesting objects as required).

Question 5 Having downloaded the text file astros.json in JSON format, you can read it into a string and parse it using the loads function from the json module. The result (that you will bind to the identifier data_astros) will be a nested Python dict.

```
In [ ]: ### GRADED
        ### Import the Python module 'json' with an appropriate import statement
        ### Load the contents of the file 'astros.json' into a string object called text_astro
        ### Use the loads function from the module json to extract the contents of the string
        ###   text_astros into a dict called data_astros
        ###
        import json
        with open('astros.json') as f:
            text_astros = f.read()
        data_astros = json.loads(text_astros)

        ### For verifying answer:
```

```
print('text_astros:\n{}\n'.format(text_astros))
print('type(data_astros): {}\n'.format(type(data_astros)))
print('data_astros:\n{}'.format(data_astros))
```

Section 1.0.1

Question 6 You've now extracted the JSON-formatted content retrieved from `api.open-notify.org` into a Python `dict` called `data_astros`. You can use standard Python techniques for working with `dict`s to extract and process data within. In particular, you will extract the number of people currently in space (associated with the key `number`) and you will construct a sorted list of their names. Don't worry about sorting by first or last names; just treat the names as a single string ans apply usual lexicographic sorting.

```
In [ ]: ### GRADED
        ### Extract the numerical value associated with the key "number" from the dict data_as
        ###    bind that value to the identifier num_people_in_space.
        ### Create a sorted list called people that contains the names of all the people
        ###    currently in space. Do not separate the names into first & last names; simply
        ###    sort the strings lexicographically as they are.
        ###
        num_people_in_space = data_astros['number']
        people = sorted(person["name"] for person in data_astros['people'])

        ### For verifying answer:
        print('Number of people currently in space: {}\n'.format(num_people_in_space))
        for person in people:
            print(person)
```

Section 1.0.1

Question 7 The URL `api.open-notify.org` also provides an API `iss-now` to determine the current location (in latitude-longitude coordinates) of the International Space Station (ISS). This changes rapidly, of course, given the speed at which the ISS travels as it orbits the earth (over 27,000 km/h). You can obtain this using `requests.get` as usual. This time, however, you'll apply the `json` method to the `Response` object returned. This transforms the object returned directly to a suitable Python object, circumventing the need to decode it into a Python string and parse it with `json.loads`. This will again yield a nested Python `dict` (that you will bind to the identifier `iss_data`).

```
In [ ]: ### GRADED
        ### Use the ISS current location API (provided as URL_ISS)) to obtain a dict
        ###    iss_data containing the ISS's current location.
        ### Use the .json method of the requests.models.Response class to parse the
        ###    response directly (rather than extracting text or dumping to a file).
        URL_ISS = 'http://api.open-notify.org/iss-now.json'

        iss_data = requests.get(URL_ISS).json()


        ### For verifying answer:
        print('iss_data.keys(): {}\n'.format(iss_data.keys()))
        for key, value in iss_data.items():
            print('key: {}\tvalue: {}'.format(key, value))
```

```
In [ ]: ### BEGIN HIDDEN TESTS
        iss_data_ = requests.get(URL_ISS).json()
        is_expected_keys = (iss_data.keys() == iss_data_.keys())
        ###
        ###
        assert is_expected_keys
        ### END HIDDEN TESTS
```

Section 1.0.1

Question 8 The values in `iss_data` are all strings (with the exception of the `timestamp` field). For practical computation involving the coordinates, it would be useful to have them represented with conventional Python floating-point values. Your task here is to construct a tuple `iss_coords` with two floating-point values: the `latitude` and `longitude` coordinates converted from the string values associated with the nested `dict` `iss_data['iss_position']`.

```
In [ ]: ### GRADED
        ### Construct a tuple iss_coords containing the (latitude, longitude) coordinates
        ### extracted from the dict iss_data.
        ###    The tuple should contain floating-point values (not strings).
        ###    Assign the tuple to the identifier iss_coords.
        ###

        iss_coords = (float(iss_data['iss_position']['latitude']), float(iss_data['iss_positio

        ### For verifying answer:
        print('iss_coords: {}'.format(iss_coords))
```

Section 1.0.1

Question 9 In many instances when querying a REST API, you want to send some sort of data in the URL's query string, i.e., you want to pass *parameters* to the REST API. If you construct the URL manually, this data would be sent as key-value pairs in the URL separated by question mark characters, e.g., https://www.example.com/get?key1=val1&key2=val2. The function `requests.get` provides a mechanism to construct the URL using the `params` keyword argument and a `dict` of strings. As an example, if you want to pass `key1=val1` and `key2=val2` to https://www.example.com/get, you would use the following code:

```
>>> base_URL = 'https://www.example.com/get'
>>> parameters = {'key1': 'val1', 'key2': 'val2'}
>>> response = requests.get(base_URL, params=parameters)
>>> response.url
https://www.example.com/get?key2=val2&key1=val1
```

This simplifies the process of sending data to a REST API in a sensible format.

In the next exercise, you will use the string `URL_ISS_PASS` as a base URL for a REST API and send the latitude-longitude coordinates of Big Ben (the clock-tower in London, UK) to retrieve a JSON response summarising the next times at which the ISS will pass over Big Ben.

```
In [ ]: ### GRADED
        ### Use the full path to ISS pass API (provided as URL_ISS_PASS)
        ###    to obtain a dict data_pass containing the pass-times of the
        ###    ISS over Big Ben.
        ###        The latitude-longitude coordinates of Big Ben are provided
        ###        in big_ben_lat & big_ben_lon respectively. Send as parameters
        ###        in the call to requests.get with the param keyword argument.
        ###    You'll have to use the .json() method to extract the data to a dict.
        ### Assign the resulting dict to data_pass.
        ###
        URL_ISS_PASS = "http://api.open-notify.org/iss-pass.json"
        big_ben_lat = 51.500809
        big_ben_lon = -0.124618
        params = dict(lat=big_ben_lat, lon=big_ben_lon)
        data_pass = requests.get(URL_ISS_PASS, params=params).json()


        ### For verifying answer:
        print(data_pass)
```

Section 1.0.1

Question 10 You now know how to use `requests` to retrieve content from remote resources on the internet. You'll switch now to working with *local files* for the rest of the assignment. You can in principle extract the contents from a `requests.model.Response` object and save that to a local file.

In this exercise, your task is to read data from the JSON file `quakes_2016_Q1.json` into a string and parse it using `json.loads`. The result will be a Python data structure containing records of all the earthquakes recorded by the United States Geological Survey (USGS) during the first quarter of 2016.

```
In [ ]: ### GRADED
        ### Use json.loads to parse the content of the file quakes_2016_Q1.json.
        ###   The location of the file is provided as PATH_quakes.
        ### Bind the result to the identifier quake_data.
        ###
        PATH_quakes = '../resource/asnlib/publicdata/quakes_2016_Q1.json'
        import json
        with open(PATH_quakes) as f:
            quake_data = json.loads(f.read())

        ### For verifying answer:
        print('type(quake_data): {}'.format(type(quake_data)))
        print('len(quake_data): {}'.format(len(quake_data)))
```

Section 1.0.1

Question 11 Having parsed the data from quakes_2016_Q1.json into a list of dictionaries, you will now do some processing. In particular, each dict in quake_data has a field 'event' which is either another dict or None. Your task, then, is: * to construct a list events containing all the nontrivial events from quake_data (i.e., filter out the None entries); * to construct a list depths by extracting the 'depth' field from all the entries in events; * to construct a list magnitudes by extracting the 'magnitude' field from all the entries in events; * to compute avg_depth, the average of all the numbers in depths; and * to compute avg_magnitude, the average of all the numbers in magnitudes.

```
In [ ]: ### GRADED
        ### Assign a list events containing the elements from quake_data
        ###    for which the 'events' field is not None.
        ### Assign a list magnitudes containing the 'magnitude' values from
        ###    all the elements of events
        ### Assign a list depths containing the 'depth' values from all the
        ###    elements of events.
        ### Compute the average value over all the entries in the list magnitudes;
        ###    assign this value to avg_magnitude
        ### Compute the average value over all the entries in the list depths;
        ###    assign this value to avg_depth
        ###
        events = [d['event'] for d in quake_data if not d['event'] is None]
        magnitudes = [d['magnitude'] for d in events]
        depths = [d['depth'] for d in events]
        avg_magnitude = sum(magnitudes) / len(magnitudes)
        avg_depth = sum(depths) / len(depths)

        ### For verifying answer:
        print('avg_magnitude: {}'.format(avg_magnitude))
        print('avg_depth: {}'.format(avg_depth))
```

Section 1.0.1

1.0.6 Using `lxml`

The `lxml` module provides useful tools for working with XML data. In particular, the function `lxml.etree.XML` parses a string formatted using valid XML and returns a Python data structure. In the final exercises, you'll use the `XML` utilities within `lxml.etree` to load data from an XML file, parse it into a suitable data structure, and run some queries.

Question 12 The first task is to parse an XML file into a data structure using the `XML` function from the module `lxml.etree`. The path to the file `cd_catalog.xml` is provided for you using the identifier `PATH_CD_XML`. Once the file is loaded into a string and parsed, you'll bind the resulting object to the identifier `catalog`.

```
In [ ]: ### GRADED
        ### Import the submodule etree directly into the global namespace from the lxml module
        ### Parse the text of the file cd_catalog.xml using etree.XML; assign the result as ca
        ###    The path is provided using the identifier PATH_CD_XML (you'll have to open the
        ###    file and read it into a string before the XML can be parsed).
        ###
        PATH_CD_XML = '../resource/asnlib/publicdata/cd_catalog.xml'

        from lxml import etree
        with open(PATH_CD_XML) as f:
            catalog = etree.XML(f.read())

        ### For verifying answer:
        print(catalog.tag, type(catalog.tag))
        print(etree.tostring(catalog).decode('utf-8'))
```

Section 1.0.1

Question 13 Having constructed an object `catalog` containing the contents of `cd_catalog.xml`, you can now start processing it. The children of the root node are all nodes with tag `CD`; each `CD` node has this structure:

```
<CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
</CD>
```

Your task is to add up the PRICE values of all the CDs in the catalog. You'll begin by extracting the PRICE fields to a list called prices. Remember, the data are all represented as strings, so you'll have to convert them to floats first (i.e., the list should contain floating-point values, not strings). Once you have a list of numbers, it's trivial to add them up.

```
In [ ]: ### GRADED
        ### Construct a list prices containing the numerical values of the PRICE fields from e
        ### Assign the sum of the entries of prices to the identifier total_prices.
        ###
        # Compute total value
        prices = [float(element.text) for element in catalog.iter("PRICE")]
        total_prices = sum(prices)

        ### For verifying answer:
        print('total_prices: {:4.2f}'.format(total_prices))
```

Section 1.0.1