SCHOOL OF INFORMATION 206: Data Oriented Programming

Beautiful National Parks

Tahmeed Tureen - Winter 2017

Abstract

This document contains a thorough explanation and guideline for the Final Project (Option #1) for the SI 206: Data-Oriented Course at the University of Michigan. The purpose of this code file is to gather a multitude of information about the National Parks and Monuments in the United States and manipulate the stripped data into an organized and readable format. The goal is that the user of this code file can access any vital or interesting information they may want to know about their interested park or monument.

Table of Contents

Limitations

There is a small number of limitations regarding this project code. These are listed as follows. I apologize for any inconveniences the limitations may have caused.

- The “Articles” on the homepage of the National Park Website have different types of nested data. This causes an overall issue because the text for some articles are under a different class while the text for others are in a different class. To get around this issue, the code file implements try/except statements. However, not all possibilities of the class names have been tried. Thus, some articles will have no descriptions associated with it. To see the code work perfectly, please use the cached file (.json) that has been attached with this .py file.
- Some Parks websites are inactive as of right now. This causes an issue because the code file is not able to strip data from an inactive page. Therefore, information about that specific park will be missing in the database. To surpass this problem, we again used the try/except method which works successfully. However, parks with inactive pages will not be added to the database.
- Running without an already cached file requires a lot of patience. The code strips data from many, many web pages and takes a long time to run (approx . However, once the data is all cached the code runs very fast (approx 0.500 to .700 seconds)

Basic Information

There were three options to choose from for the final project in this course, I chose the "Beautiful Soup on National Parks" Project (Option #1). The goal of this option is to strip data from the National Parks website using BeautifulSoup. The website page is: https://www.nps.gov/index.htm

This code file will create a database that includes three specific tables: Parks, States and Articles. These are all thoroughly discussed later in this document.

This code file can be used to find interesting information about the national parks/monuments in the United States. The advantage of using this code file versus using the website is that the code will strip all of the necessary data and save it to a personal computer allowing for access without internet or data service. Another advantage is that the code file will systematically collect and preserve data regarding specifics into a database which can be easily accessed using any data browser for SQLite instead of clicking on a website several times.

To RUN this code, a user must either have terminal on Mac or GitBash on Windows. After opening up the program, the user must enter the correct directory/repository and run the code file with the command "python 206_beautiful_national_parks.py" (Note that you may have saved the file with a different name). After the code is run, it will create a cached file, a database and give the user the adequate information that is specified later in this document.

Prior To Running Code

To successfully run this code the following must be imported (which are all included in the .py file)

- Unittest, codecs, BeautifulSoup, json, requests, sqlite3, collections, sys
- Some of these will require a pip install which a user can easily find directions for by googling "pip install (module he/she is interested in)"

- There are no specific files that is needed to run this code, however, a sample cached file (si206final_project_cache.json) is included if for any reason the National Park Website or other websites are currently down.

**For Windows Users, this line is a necessity otherwise the code will not run
sys.stdout = codecs.getwriter('utf-8')(sys.stdout.buffer)

Files Included

Several files are included on the GitHub repository or zip file. These are as follows:

- 206_beautiful_national_parks.py (The entire code file)
- si206final_project_cache.json (The original cached file that was used to create the code)
- finalproject_database.db (The database that contains the original data regarding the parks/monuments, articles and states)
- Sample_summary_output.pdf (PDF of a sample output of the code)

Functions

Below you will find an extensive description of each function in the code file.

- get_states_data()
    - Input: None
    - Return Value: A list of urls for each park/monument webpage (stateslink_list)
    - Behavior: This function will systematically access the root page of the National Parks website and go to each state's webpage and strip the url for the state webpage. This webpage includes all of the parks that are in that state. The stripped urls are then appended to a list and saved to a cache file in the code repository.
- get_parks_data()
    - Input: None
    - Return Value: An extensive list of HTML strings which each represent a park/monument 's webpage

    - Behavior: This function will use the get_states_data() function and access each state's webpage and go through each of the park/monument's webpages and strip that data. The data is then stored as a list of HTML strings which is saved in a cached file.
- get_articles_data()
    - Input: None
    - Return Value: A small list of HTML strings which represent the webpage of the articles on the homepage of the National Parks website.
    - Behavior: Similar to the get_parks_data() function, this function systematically accesses the web pages of the articles on the homepage of the website and strips the HTML string of those pages and appends it to a list. This is also saved into a cached file.

      *Note: Only the articles on the first row of the homepage is collected. This is because some of the articles below the first row are actually not articles, but are information pages about the National Parks website.
- get_avg_temp()
    - Input: None
    - Return: A dictionary with key-value pair that is a U.S. State and it's average weather in degrees Celsius.
    - Behavior: This function accesses the website: https://www.currentresults.com/Weather/US/average-annual-state-temperatures.php and strips data from a table which includes the U.S. States and its respective weather. This function specifically only strips the weather in degrees Celsius. Finally, the data is saved to a cache file.

Classes

Below you will find an extensive description of each class in the code file.

- NationalPark(object)
    - Class Variables: A dictionary that has key-value pair of U.S. state abbreviation and the full state name.

- Input: An HTML string input, this input must represent a webpage of a national park or monument. Otherwise, it will not work. This is a REQUIRED INPUT
- There are four instance variables under the constructor.
  - self.park_soup - The soup object of the inserted HTML
  - self.name - Name of the park/monument
  - self.states - list of all state(s) in which the park is.
  - Self.tup_links - a list of tuples with the first element of the tuple being a link to a certain page and the second element is the name of that page. For example: (havo/planyourvisit/index.htm, Plan a Visit), (havo/planyourvisit/basicinfo.htm, Basic Info) etc.
- There are six methods in this class.
  - __str__(self) - This method simply models the printing format of the NationalPark Class. For example, if an instance of this class is printed it will print "Katmai is located in Hawaii"
  - get_planpage(self) - Returns the string description of the park from its respective website
  - get_basicinfo(self) - Returns the url link of the park's basic info webpage
  - get_faqs(self) - Returns the url link of the faqs page of the park. If such a web page does not exist for a park, then a random page link will be returned (This is because some parks don't have faqs pages)
  - modify_state(self) - This method modifies the self.states instance variable. Some parks are located in multiple different states. To get around this issue, this method can be implemented to assign only the first state in the list of states in which the park resides. Does not return anything ** This may cause some users confusion.

- Article(object)
  - The REQUIRED INPUT is a HTML string that represents a webpage of an article from the home page of the National Parks website. The code will not run without this input for this class.
  - This class has three instance variables:
    - self.article_soupx

        - self.name - The title of the article. If the title does not exist or has a unique HTML nest, then this variable will be assigned this value: “TITLE IS N/A due to Unique HTML Nest”
        - self.text - The description or writing of the article. If the description does not exist or has a unique HTML nest (different from the general code), then this variable will be assigned this value: “Description N/A due to Unique HTML Nest”
    - This class has only one method:
        - __str__() - This method simply models the printing format of the Article Class. For example, if an instance of this class is printed it will print “The title of the article is “Title Name” and the description is: “Description” ”

Database Creation

As mentioned earlier in this document, this code file will create a database with three distinct tables. These tables are extensively described below:

- Parks
    - Each row of the table represent a national park or monument
    - The attributes of each park/monument are:
        - Name (Primary Key) - The name of the park/monument
        - Description - A text description of the park. For example, the history of the park or its current conditions (different for each park)
        - State - The state in which the park resides. Only one state is contained in this column (Refer to NationalPark Class Documentation above)
        - PlanVisit - URL link to the plan a visit webpage for the park
        - BasicInfo - URL link to the basic info webpage for the park
        - FAQsMISC - URL link to the FAQs webpage for the park. If the FAQs page does not exist for a specific park, then a random page will be in this column. This is because some parks do not have a FAQS webpage.

- Articles
    - Each row represents an article from the home page of the National Park website
    - The attributes of each article are:
        - Name - The name of the article (Note: Some articles don't have appropriate names)
        - Description - The writing/description of the article. Some are long and some are short (Note: Some articles may not have an appropriate description)

** THERE ARE ONLY TWO ARTICLES IN THE DATABASE TABLE, PLEASE REFER TO THE get_articles_data() function documentation.

- States
    - Each row represents a U.S. state (There are 50 states in the table here)
    - The attributes of each state are:
        - Name - Name of the State. (Example: Michigan, California)
        - Abbreviation - Two letter abbreviation of the state (Example: MI, CA)
        - Weather - The average weather of the state in degrees celsius.

Data Manipulation Code:

There were several approaches done to manipulate the stripped data.

Python Tools Used:

- ★ List Comprehensions
- ★ Set Comprehensions
- ★ Counter from collections library
- ★ most_common() from collections library
- ★ Sorting with a key parameter (Lambda function)
- ★ Dictionary Accumulation of counts

Data Processing #1

The first manipulation was done by using an SQL query and getting a list of all of the parks

that have average weather greater than 60 degrees celsius. This was done by utilizing the INNER JOIN method in the SQL query and combining data from both the States table and the Parks table. List comprehension is used to accumulate all of the states and their an output is printed on git bash/terminal. The summary of findings for the original cached data can be found at the end of this documentation or in the attached summary.pdf file.

Data Processing #2

The 2nd manipulation was done using set comprehension and splitting each of the descriptions for the articles using the split() method. Before the set was created, a SQL query was made to get all of the descriptions of the articles in the Articles Database. The set included all of the words that the articles mentioned in its description. This set was then used to check whether or not the following parks were mentioned in the articles: "Katmai", "Pullman", "Keweenaw", "Appalachian", "Saratoga", "Pipestone", "Yellowstone", "Christiansted". The output is also printed on gitbash/terminal when the code is run. The summary of findings for the original cached data can be found in the attached summary.pdf file.

Data Processing #3

The 3rd manipulation was done using the collections library and the collections library. It uses the same query that was used in Data Processing #2. First a collections counter instance is created which then uses dictionary accumulation to create a dictionary with each word from the article descriptions and their respective frequency. Then the sorted function is utilized using a lambda key function which will order the words in alphabetical order if for any reason there is a count tie. Some common stoppage words like “the”, “is” etc. have been omitted for the counts. The summary of findings for the original cached data can be found in the attached summary.pdf file.

Data Processing #4

The fourth manipulation is exactly the same as Data Processing #4 but it uses the SQL query that gets all of the descriptions of each park in the database. The summary of findings for the original cached data can be found in the attached summary.pdf file.

Data Processing #5

The fifth and final manipulation uses the SQL query that gets the states and their count of parks. The query is an INNER JOIN that is applied to the States and Parks tables. The collections library is used again along with the Counter() method which creates a dictionary of key value pair State and their count of parks. Finally, the most_common() method is used to get the state with the highest number of parks. The output includes the dictionary and a statement that tells the user which state has the most parks. The summary of findings for the original cached data can be found in the attached summary.pdf file.

Reason for Code:

Why did I choose to do this project?
As a statistics major at the University of Michigan, I've recently recognized the importance of a knowledge of coding and data structures especially in the languages of R and Python. I chose to take this class to get a more fundamental exposure to Python. In statistics, we usually deal with large data sets and often times the data is given to us. What this project helped me accomplish was honing my skills in data stripping as well as data manipulation using BeautifulSoup and the requests module. I believe that this project has helped me better understand how to collect data from websites and organize it in a way that works best for me. In addition, I will be taking SI 330: Data Manipulation in the upcoming semester so I thought getting more comfortable with BeautifulSoup, requests, and SQL will be very helpful before making the transition to a 300 level SI course.

All in all, I believe that this course has been a wonderful journey. I've learned a lot and have already applied some of the skills I learned to other classes and my research projects. Therefore, I am grateful to the School of Information for allowing an LS&A student as myself to enroll in such an informative course. Finally , I would like to thank Jackie Cohen and Mauli Pandey for helping me throughout this semester and answering the tremendous amount of questions that I had.

Relevant Lines of Code:

- Line(s) on which each of your data gathering functions begin(s): ~ 54, 87, 136, 354
- Line(s) on which your class definition(s) begin(s): ~ 180, 297
- Line(s) where your database is created in the program: ~ 400-431
- Line(s) of code that load data into your database: ~ 442-582
- Line(s) of code where your data processing code occurs: ~ 585+
- Line(s) of code that generate the output: ~ 585+ (All print statements generate output that can be seen in terminal or gitbash)