**Project 2 - Scraping and Crawling nps.gov**

You will create a program to scrape and search for information about National Sites (Parks, Heritage Sites, Trails, and other entities) from <https://www.nps.gov>. You will also add the ability to look up nearby places using the [MapQuest API](https://developer.mapquest.com/).

**Knowledge Required**

The contents of each part are covered in lecture as follows:

* Part 1 & Part 2 (Scraping): In the week of 18th February.
* Part 3 (Crawling): In the week of 25th February.
* Part 4 (API & caching): Already covered.
* Part 5 (Interactive interface): Already covered.

So, one strategy is to implement Part 1 & Part 2 in the week of 18th February, and implement other parts after the week of 25th February.

**Get started**

You will clone the starter code and the test code from GitHub by accepting the GitHub Classroom assignment, as described below.

GitHub link: <https://classroom.github.com/a/DB4cpZh8>

Hint: git clone https://xxxx

Starter code: proj2\_nps.py

Test file: proj2\_nps\_test.py

**What to submit**

Accept the GitHub Classroom assignment invitation. This will create a GitHub repo in your personal account that you will use to track and submit your project. Clone the repo and work on it, committing and pushing as you go. Make sure to push your final submission to the GitHub repo before deadline, and submit the repo URL on Canvas (this is for backup). In addition, since you need to push only proj2\_nsp.py to GitHub, use .gitignore to exclude unnecessary files. Make sure to create your .gitignore *before* adding (or generating) any new files, such as your cache file (e.g., cache.json), \_\_pycache\_\_, or secrets.py.

Also, please observe the following:

* Since we will grade both with and without cache, do not forget to test your solution code with and without a cache file. (When you want to run without a cache, delete the cache file and run.) Do *not* push your cache file to GitHub.
* Use the test file (run python/python3 proj2\_nps\_test.py) for testing Part 1, 2, 3 and 4 of your project.
* You can add any functions with docstrings but must leave the existing functions, including names and parameters, as they are. This is required for testing.
* Do not change the name of the file proj2\_nps.py
* Do not change any of the contents of the file proj2\_nps\_test.py
  + You can create other files, including other test files, if you would like, but you may not change this file or rename the main program file.

Failure to follow these guidelines may result in point deductions.

**Part 1: Scrape state URLs**

In part 1, you will scrape the page <https://www.nps.gov/index.htm> with the goal of being able to make a dictionary that maps state names to state page URLs.

The dictionary keys will be the state name (**in lower case**),  and the value will be the URL for the state’s page on nps.gov.

(e.g. {'michigan': 'https://www.nps.gov/state/mi/index.htm', ...} ).

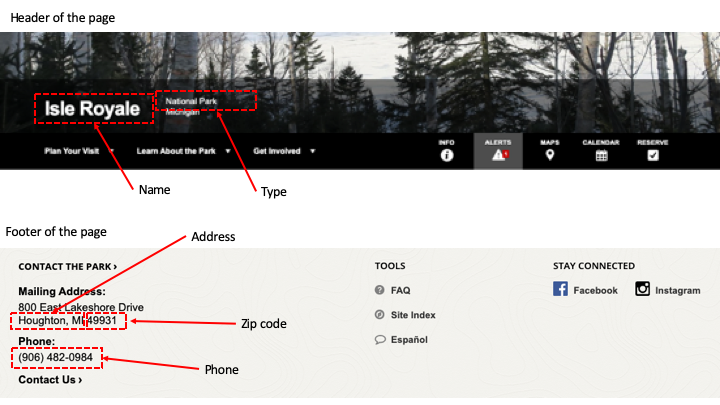
This dictionary should include all states listed on <https://www.nps.gov/index.htm>. The links to state pages can be accessed from the dropdown box under the label “FIND A PARK” (this is a clue to help you find the part of the page you will need to extract):



To pass the included tests, you will need to complete the implementation of build\_state\_url\_dict() to return the correct dictionary.

**Part 2: Create an instance of a national site**

In part 2, you will scrape individual site pages (e.g. <https://www.nps.gov/isro/index.htm> or <https://www.nps.gov/yell/index.htm>) with the goal of being able to create instances of NationalSite. Each NationalSite (instance) should have attributes on name, category (e.g., ‘National Park,’ ‘National Monument’, or blank), address, zip code, and phone number. The required attributes for the NationalSite class can be seen in the starter code file in detail.



NationalSite class should have a method info() that returns a string representation of itself.

The format is <name> (<category>): <address> <zip> .

Example: Isle Royale (National Park): Houghton, MI 49931

To pass the included tests, you will need to complete the implementation of get\_site\_instance(site\_url) to return a NationalSite object.

**Part 3: Crawling**

In part 3, you will *crawl* nps.gov with the goal of being able to print out information about any National Site listed on the site, organized by state. The information will include name, category, and mailing address. You will use functions that you implemented in Part 1 (build\_state\_url\_dict()) and Part 2 (get\_site\_instance()) to achieve the goal of Part 3.

First, your program will ask a user to input a state name (case-insensitive).

Second, to pass the included tests, you will need to edit the function in the starter code get\_sites\_for\_state(state\_url) that takes a state page URL (e.g. “<https://www.nps.gov/state/az/index.htm>”) and returns a list of NationalSite objects in the state page.

Third, based on the returned value from get\_sites\_for\_state(state\_url), print national sites in the state in the following format: [number] <name> (<type>): <address> <zip code>.

Example : [1] Isle Royale (National Park): Houghton, MI 49931

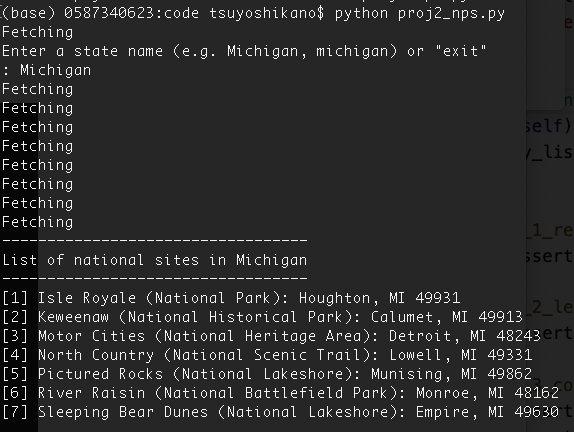
Finally, implement caching so that you only have to visit each URL within nps.gov once (and subsequent attempts to visit, say  <https://www.nps.gov/index.htm> or <https://www.nps.gov/state/mi/index.htm> or <https://www.nps.gov/isro/index.htm> are satisfied using the cache rather than another HTTP request). Print Using Cache when you use cache data, and print Fetching  when you make a HTTP request. (This will also have the side effect of dramatically speeding up your development time!)

**Note**

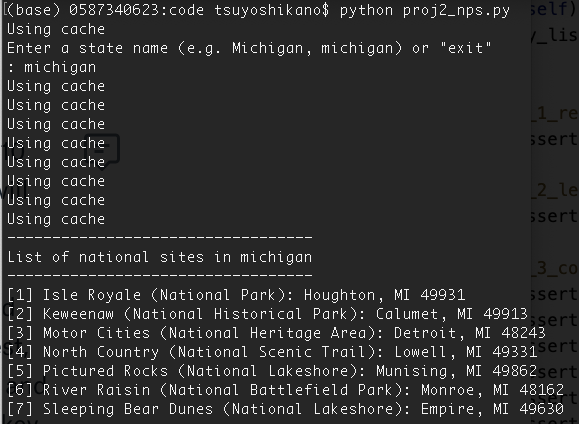
* Since you need to access multiple pages to create an object list, you are likely to print Using Cache or Fetching multiple times.
* You also need to modify functions in Part 1 and Part 2 to use caching.

**Sample outputs**

**Case 1: Run without cache (upon first access)**



**Case 2: Run with cache (upon subsequent accesses)**



**Part 4: Find nearby places**

Implement a function get\_nearby\_places(site\_object) that finds up to 10 places in or near the zip code of the national site’s mailing address. The function will return a converted python dictionary from the MapQuest API.

In order to implement API request, go to [MapQuest API documentation](https://developer.mapquest.com/documentation/search-api/v2/radius-search/get/) and find out the necessary URL and parameters. In order to make an API request, you need to obtain API key from [MapQuest](https://developer.mapquest.com/) (click “**Get your free API Key**”). Once you get an API key, make secrets.py in the same folder with your project code and save the API key in secrets.py.

This is an example of secrets.py. (Replace “xxxxxxx” with your API key.)

API\_KEY = "xxxxxxx"

For a MapQuest API request, you need to use the following five parameters.

* key: API key from secrets.py
* origin: Zip code of a national site (Use NationalSite instance attribute.)
* radius: Distance from the origin to search is 10 miles.
* maxMatches: The number of results returned in the response is 10.
* ambiguities:  “ignore”
* outFormat: “json”

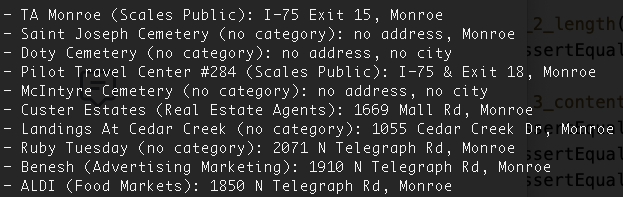
Based on the returned value from get\_nearby\_places(site\_object), print up to 10 places in the following format: - <name> (<category>): <street address>, <city name> . If a place doesn’t have code or address, display “no category,” as shown in Example 2.

Example 1 : - ALDI (Food Markets): 1850 N Telegraph Rd, Monroe

Example 2: - McIntyre Cemetery (no category): no address, no city

Finally, implement caching so that you only have to call the API once each zip code. Print Using Cache when you use cache data, and print Fetching  when you make an API request.

**Sample output**



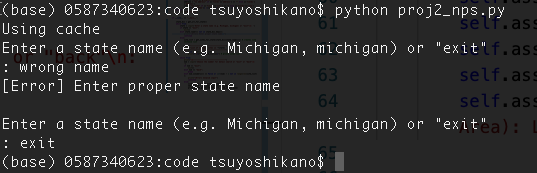
**Part 5: Create an interactive search interface**

For this last part, you will add the ability for users to enter their own queries and receive nicely formatted results. The steps of the interactive search interface are as follows.

[**Step 1]**

Ask a user to enter a state name (case-insensitive).  If a user enters “exit”, end the program. If a user enters an invalid state name, print an error and ask the user to input again.

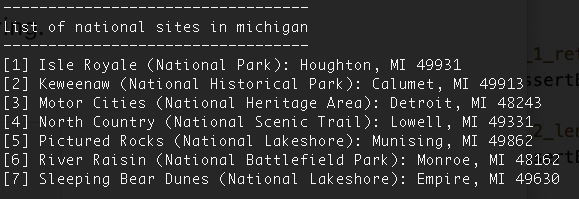
Sample output of wrong input and exit



**[Step 2]**

When a user inputs a proper state name, print a list of national sites in the state with a proper header (List of national sites in <state>) and numbering.

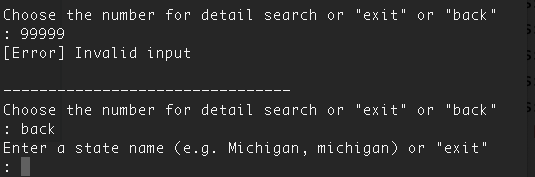
Sample output (when a user enters “michigan”)



**[Step 3]**

Ask a user to enter a number in the list of Step 2. If a user enters “back”, go to Step 1. If a user enters “exit”, end the program. If a user enters an invalid number, print an error and ask the user to input again.

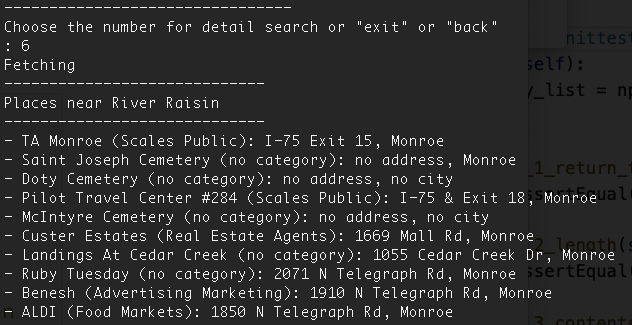
Sample output (an error case and “back” case)



**[Step 4]**

When a user inputs a proper number, print a list of (up to 10) nearby places with a header (Places near <national site>).

Sample output (when a user enters “michigan”, then “6”)



**[Step 5]**

Repeat Step 3 and 4 until the user enter “exit”.

**Rubrics**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Req | Part | Description | Category | Point Value |
| 1 | 1 | Function build\_state\_url\_dict returns a dictionary.  (Whether the code passes test\_1\_1\_return\_type or not) | Code | 5 |
| 2 | 1 | Function build\_state\_url\_dict returns a dictionary that covers all the states listed in nps.gov. (56 items)  (Whether the code passes test\_1\_2\_return\_length or not) | Code | 10 |
| 3 | 1 | Function build\_state\_url\_dict returns a dictionary that maps state names to state page URLs. (e.g. {'michigan': 'https://www.nps.gov/state/mi/index.htm', ...} ).  (Whether the code passes test\_1\_3\_contents or not) | Code | 10 |
| 4 | 2 | Function get\_site\_instance returns an instance of NationalSite that has proper name and category attributes.  (Whether the code passes test\_2\_1\_basic or not) | Code | 10 |
| 5 | 2 | Function get\_site\_instance returns an instance of NationalSite that has proper address and zipcode attributes.  (Whether the code passes test\_2\_2\_address or not) | Code | 10 |
| 6 | 2 | Function get\_site\_instance returns an instance of NationalSite that has proper phone attribute.  (Whether the code passes test\_2\_3\_phone or not) | Code | 10 |
| 7 | 2 | Function get\_site\_instance returns an instance of NationalSite that has a method info() to print the site information.  (e.g. Isle Royale (National Park): Houghton, MI 49931)  (Whether the code passes test\_2\_4\_str or not) | Code | 10 |
| 8 | 3 | Function get\_sites\_for\_state returns a list  (Whether the code passes test\_3\_1\_return\_type or not) | Code | 5 |
| 9 | 3 | Function get\_sites\_for\_state returns a list with proper length. (e.g. Michigan has 7 sites and Wyoming has 10 sites.)  (Whether the code passes test\_3\_2\_length or not) | Code | 10 |
| 10 | 3 | Function get\_sites\_for\_state returns a list of NationalSite instances. And the instance has proper attributes (i.e. name, category, address, zipcode, and phone).  (Whether the code passes test\_3\_3\_contents or not) | Code | 10 |
| 11 | 4 | Function get\_nearby\_places returns a dictionary  (Whether the code passes test\_4\_1\_basic or not) | Code | 10 |
| 12 | 4 | Function get\_nearby\_places returns a dictionary with the contents of a MapQuest API response that has keys resultCount and options with proper values.  (Whether the code passes test\_4\_2\_contents or not) | Code | 10 |
| 13 | 5 | [step 1] When a user inputs an invalid state name, print an error message and ask the user to input again. | Behavior | 5 |
| 14 | 5 | [step 1] When a user inputs “exit”, end the program. | Behavior | 5 |
| 15 | 5 | [step 2] When a user input a valid state name, print a header with a state name, such as List of national sites in michigan | Behavior | 5 |
| 16 | 5 | [step 2] When a user input a valid state name, print a list of national sites in any format. | Behavior | 5 |
| 17 | 5 | [step 2] When a user input a state name, prints a list with the following format.  [number] <name> (<type>): <address> <zip>  e.g. [1] Isle Royale (National Park): Houghton, MI 49931 | Behavior | 5 |
| 18 | 5 | [step 2] Prints Using cache or Fetching appropriately, depending on whether the state has been accessed previously. | Behavior | 10 |
| 19 | 5 | [step 3] When a user inputs an invalid number, print an error and ask the user to input again | Behavior | 5 |
| 20 | 5 | [step 3] When a user inputs “exit”, end the program. | Behavior | 5 |
| 21 | 5 | [step 3] When a user inputs “back”, go back to step 1 (choose a state), and new state search works. | Behavior | 5 |
| 22 | 5 | [step 4] When a user inputs a valid number, to choose a National Site, print header with a site name, such as  Places near Isle Royale. | Behavior | 5 |
| 23 | 5 | [step 4] When a user inputs a valid number, print nearby place list (probably 10 lines but may be less in some cases). | Behavior | 5 |
| 24 | 5 | [step 4] When a user inputs a valid number, print nearby place list with the following format.  - <name> (<category>): <street address>, <city name>  e.g. - BP Station (Gas Stations): ST HWY 26, Houghton | Behavior | 5 |
| 25 | 5 | [step 4] Prints Using cache or Fetching appropriately, depending on whether the API request has been accessed previously. | Behavior | 10 |
| 26 | 5 | [step 5] After step 4, go to step 3 (ask for a National Site number again) and it works. | Behavior | 5 |
| 27 | ALL | Submitted GitHub repo has only pj2\_nps.py and pj2\_nps\_test.py.  Or, the repo has .gitignore that mentions other unnecessary files (such as secrets.py, a cache file, and \_\_pycache\_\_).  (The second choice is for students who mistakenly pushed other files. You don’t need to delete them from your GitHub repo.) | Behavior | 5 |
| 28 | ALL | Well-constructed code that follows our guidelines. | Code | 5 |
|  |  | **Total** |  | 200 |