2020 SI 507 Waiver Test: Scraping and Crawling nps.gov

Important note

To be considered for the SI 507 waiver, you must first complete the Google sign-up form. If you do not complete this form, you will not be considered for the waiver.

Test assignment summary

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.

Getting started

You can download the starter files for the test assignment here:

Starter code: si507_waiver.py
Test file: si507_waiver_test.py

What to submit

The only file you need to submit to Canvas is your copy of the `si507_waiver.py` file. You do not need to submit the test file (we have it) or your `secrets.py` file, since we'll run the code with our own API key.

Also, please observe the following:

- Please write your code in **Python 3** (the current version is 3.8.5). Python 2.7 has been deprecated for quite a while, and submissions that require us to run your program using a python 2 interpreter will not be accepted. If you are a Mac user, this means you will need to install (if you don't have it already) python 3 from python.org. The pre-installed python is version 2.7, which is not what you want for this assignment (or life in general).
- 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* submit your cache file!

- Before submitting, use the test file (run, on Windows or Linux, `python si507_waiver_test.py` or, on the Mac, `python3 si507_waiver_test.py`) to test 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 specified. This is required for testing.
- Do not change the name of the file `si507_waiver.py` since that's the name that the test file expects.
- Do not change any of the contents of the file `si507_waiver_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. In addition, if you decide to break your code into multiple program files, submit those files along with `si507_waiver.py` in Canvas. Make sure your main program file, `si507_waiver.py`, correctly imports additional program files. For simplicity, though, you may just want to stick with writing all your code in the `si507_waiver.py` 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.

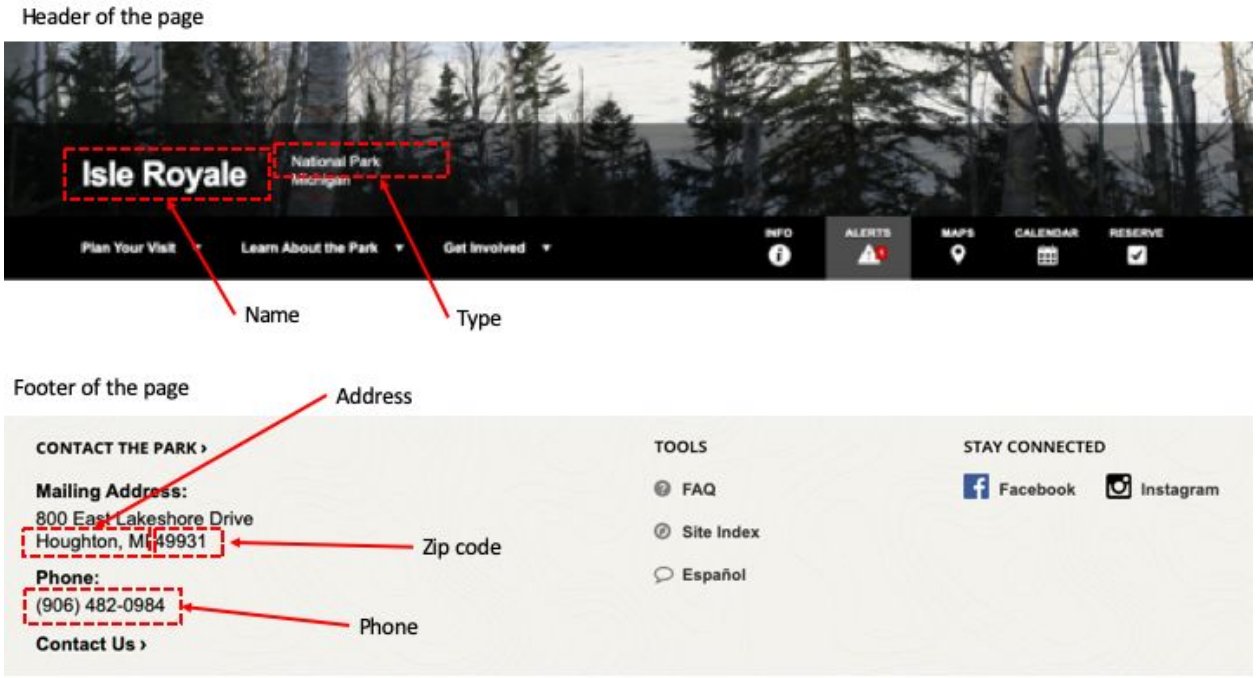
The dictionary keys will be the state name (**in lower case**), and the value will be the URL for the state's page on `nps.gov`.
(e.g. `{'michigan': 'https://www.nps.gov/state/mi/index.htm', ...}`).

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)

```
(base) 0587340623:code tsuyoshikano$ python proj2_nps.py
Fetching
Enter a state name (e.g. Michigan, michigan) or "exit"
: Michigan
Fetching
Fetching
Fetching
Fetching
Fetching
Fetching
Fetching
Fetching
---------------------------------
List of national sites in Michigan
---------------------------------
[1] Isle Royale (National Park): Houghton, MI 49931
[2] Keweenaw (National Historical Park): Calumet, MI 49913
[3] Motor Cities (National Heritage Area): Detroit, MI 48243
[4] North Country (National Scenic Trail): Lowell, MI 49331
[5] Pictured Rocks (National Lakeshore): Munising, MI 49862
[6] River Raisin (National Battlefield Park): Monroe, MI 48162
[7] Sleeping Bear Dunes (National Lakeshore): Empire, MI 49630
```

Case 2: Run with cache (upon subsequent accesses)

```
(base) 0587340623:code tsuyoshikano$ python proj2_nps.py
Using cache
Enter a state name (e.g. Michigan, michigan) or "exit"
: michigan
Using cache
Using cache
Using cache
Using cache
Using cache
Using cache
Using cache
Using cache
---------------------------------
List of national sites in michigan
---------------------------------
[1] Isle Royale (National Park): Houghton, MI 49931
[2] Keweenaw (National Historical Park): Calumet, MI 49913
[3] Motor Cities (National Heritage Area): Detroit, MI 48243
[4] North Country (National Scenic Trail): Lowell, MI 49331
[5] Pictured Rocks (National Lakeshore): Munising, MI 49862
[6] River Raisin (National Battlefield Park): Monroe, MI 48162
[7] Sleeping Bear Dunes (National Lakeshore): Empire, MI 49630
```

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 dictionary based on the output from the MapQuest API.

In order to implement the API request, go to MapQuest API documentation and find out the necessary URL and parameters. In order to make API requests, you need to get an API key from MapQuest (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`.

`Secrets.py` file should contain a single line like this (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 data you get from Mapquest, `get_nearby_places(site_object)` should print out a list of 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 for each zip code. Print `Using Cache` when you are using cache data, and print `Fetching` when you make an API request.

Sample output of the get_nearby_place function

```
- TA Monroe (Scales Public): I-75 Exit 15, Monroe
- Saint Joseph Cemetery (no category): no address, Monroe
- Doty Cemetery (no category): no address, no city
- Pilot Travel Center #284 (Scales Public): I-75 & Exit 18, Monroe
- McIntyre Cemetery (no category): no address, no city
- Custer Estates (Real Estate Agents): 1669 Mall Rd, Monroe
- Landings At Cedar Creek (no category): 1055 Cedar Creek Dr, Monroe
- Ruby Tuesday (no category): 2071 N Telegraph Rd, Monroe
- Benesh (Advertising Marketing): 1910 N Telegraph Rd, Monroe
- ALDI (Food Markets): 1850 N Telegraph Rd, Monroe
```

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]**
When the program starts, ask the user to enter a state name (case-insensitive). If the user enters “exit”, the program should quit. If a user enters an invalid state name, print an error and ask the user to enter a state name again.

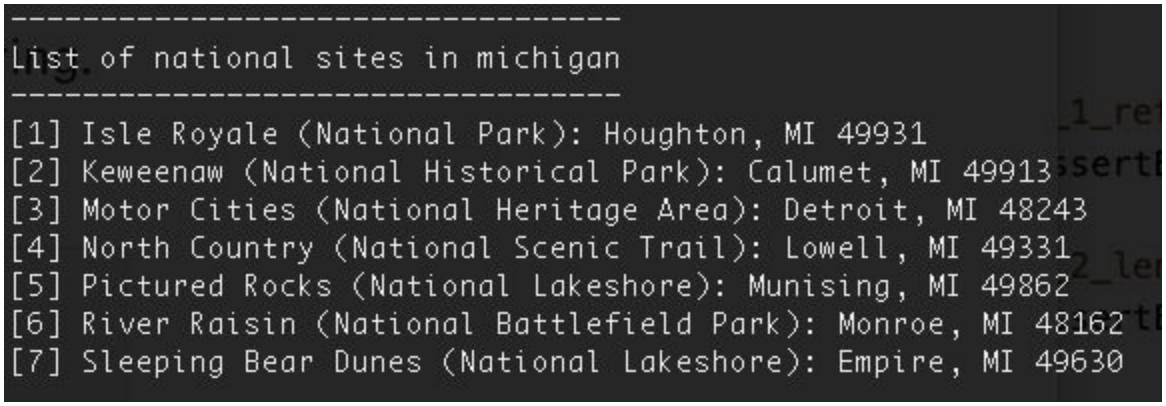
Sample output of wrong input and exit

```
(base) 0587340623:code tsuyoshikano$ python proj2_nps.py
Using cache
Enter a state name (e.g. Michigan, michigan) or "exit"
: wrong name
[Error] Enter proper state name

Enter a state name (e.g. Michigan, michigan) or "exit"
: exit
(base) 0587340623:code tsuyoshikano$
```

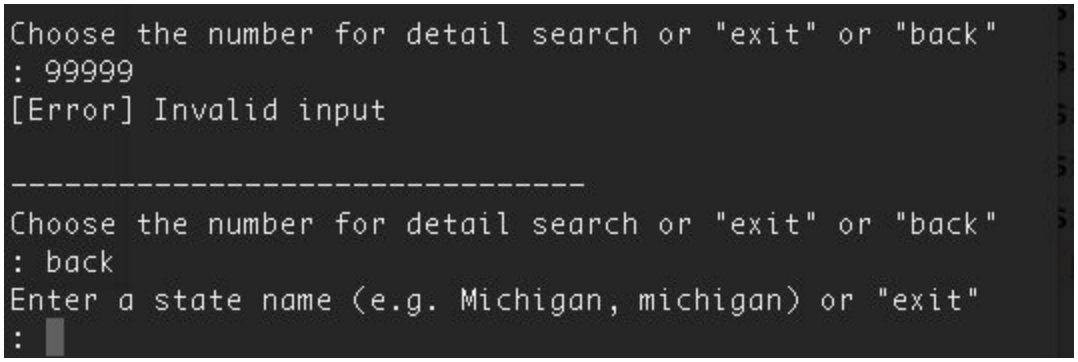
[Step 2]
When a user inputs a valid state name, print a list of national sites in the state with a nicely formatted header (`List of national sites in <state>`) and numbering.

Sample output when a user enters “michigan”:



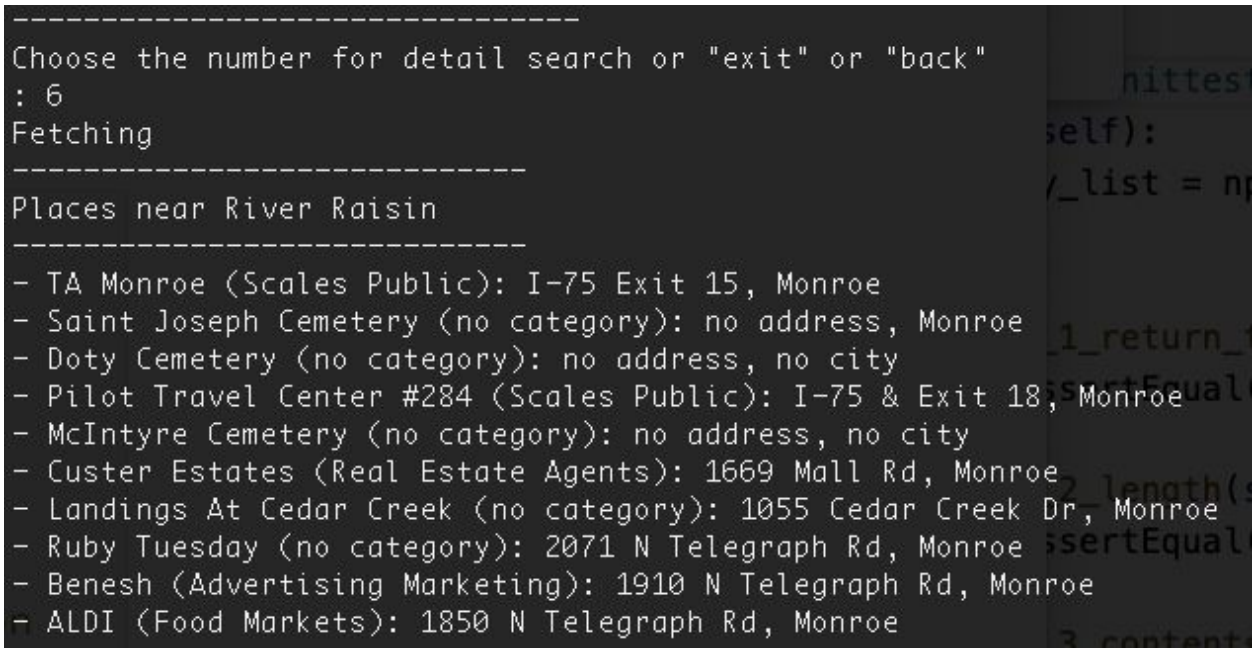
[Step 3]
After the list of national sites is printed, offer to the user to enter a number from the list to find nearby places, or to enter “back” to search national sites for another state. If the user enters “exit”, end the program. If the user enters an invalid number, print an error message and ask the user to try again.

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



[Step 4]
When a user enters a valid number, print a list of (up to 10) nearby places with a header formatted as `Places near <national site>`.

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



[Step 5]
Repeat Step 3 and 4 until the user enters “exit”.

Grading Rubric and Requirements for Passing the Waiver Test

Your code can earn up to 200 points. **To pass the waiver test, you will need to earn 90% or 180 points.** The grading rubric will be as follows:

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`. (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 a 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 a nearby place list in 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	Well-constructed code that follows our guidelines.	Code	10
		Total		200