SI 507 Project 2

# Project Overview

You will create a program to scrape and search information about National Sites (Parks, Heritage Sites, Trails, and other entities) from nps.gov. You will also add the ability to look up nearby places using the Google Places API and to display National Sites and Nearby Places on a map using plotly.

Starter code:

[proj2\_nps.py](https://www.dropbox.com/s/64fldvjy2pl8a7g/proj2_nps.py?dl=0)

[secrets.py](https://www.dropbox.com/s/kfr98sy7vwygxw7/secrets.py?dl=0)

Test file:

[proj2\_nps\_test.py](https://www.dropbox.com/s/ae7hsrol1sen73x/proj2_nps_test.py?dl=0)

\* Part of your project will be graded using this test file.

You only need to submit proj2\_nps.py file.

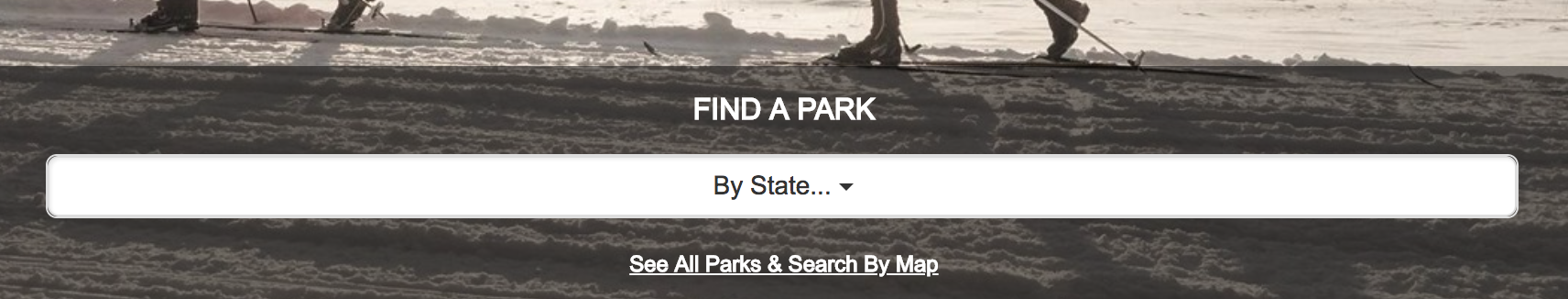
Also please observe the following:

* 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 (80 points)

In part 1 you will scrape nps.gov with the goal of being able to print out information about any National Site listed on the site, organized by state. Information will include the site name, site type, and the physical (or mailing) address. Your program will start crawling at <https://www.nps.gov/index.htm>, and from there crawl pages for particular states and then pages for particular sites. The links to state pages can be accessed from the dropdown box labeled “FIND A PARK.”



To pass the included tests, you will need to edit the function in the starter code get\_sites\_for\_state(state\_abbr) that takes a state abbreviation and returns a list of NationalSites that are in that state. The required attributes for the NationalSite class can be seen in the skeleton code file (proj2\_nps.py).

At the basic level, each NationalSite (instance) should be created with a name, type (e.g., ‘National Park,’ ‘National Monument’, ‘National Historic Site’), and description. All of these can be found on the landing page for a particular state (e.g., <https://www.nps.gov/state/mi/index.htm>).

In addition, you should visit the detail page for each site to extract additional information--in particular the physical address of the site. To do this, you will have to crawl one level deeper into the site, and extract information from the site-specific pages (e.g., <https://www.nps.gov/isro/index.htm>).

Printing a NationalSite object should return a string representation of itself (using \_\_str\_\_( )) of the following form: <name> (<type>): <address string>

For example:

Isle Royale (National Park): 800 East Lakeshore Drive, Houghton, MI 49931

Finally, though you should really consider doing this first to dramatically speed up your development time, 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/state/mi/index.htm> or <https://www.nps.gov/isro/index.htm> are satisfied using the cache rather than another HTTP request).

Grading

* [30 points] Implement basic searching by state and creation of NationalSites with name and type. Pass TestStateSearch.test\_basic\_search( )
* [30 points] Implement adding address information to NationalSites by crawling. Pass TestStateSearch.test\_addresses( )
* [10 points] Implement \_\_str\_\_( ) as specified. Pass TestStateSearch.test\_str( )
* [10 points] Add caching so that you never have to visit a page on the NPS site more than once.

# Part 2 (40 points)

Implement a function get\_nearby\_places(site\_object) that looks up a site by name using the Google Places API and returns a list of up to 20 nearby places, where “nearby” is defined as within 10km (note: 20 results is the default maximum number returned by the Google Places API without paging).

Getting the list of nearby places will require two calls to the google API: one to get the GPS coordinates for a site (tip: do a text search for <site.name> <site.type> to ensure a more precise match--it turns out there are lots of places called “Death Valley” that aren’t National Parks!), and another one to get the nearby places. Documentation on the Google Places API can be found here: <https://developers.google.com/places/web-service/search>.

You will need to get a Google API key following instructions [here](https://www.dropbox.com/s/uhbhueweyc3eq06/Google%20Place%20API%20Instruction.docx?dl=0). Implementing caching for this portion of the project is STRONGLY recommended.

get\_nearby\_places(site\_object)should return a list of NearbyPlace objects.

At a minimum, a NearbyPlace needs to have the name of the place as an attribute. You may find it useful to add other attributes as well. A NearbyPlace should include a \_\_str\_\_( ) method, which simply prints the Place name.

Note:

* If you do a search on a NationalSite using <site name> <site type> (e.g., “Death Valley National Park” or “Motor Cities National Heritage Area”) and Google Places does not return any results (or returns results, but none of them have the specific name you searched for), your list of “Nearby Places” should be an empty list.

Grading:

* [30 points] Return a list of up to 20 NearbyPlaces from get\_nearby\_places(site\_object). Each place has a properly configured name attribute. Pass test\_nearby\_search( ).
* [10 points] adding caching so that you only do a particular nearby search once (items in your cache don’t need to expire, even though technically the data could change given enough time.)

# Part 3 (40 Points)

Use plotly to display maps of NationalSites within a state and NearbyPlaces near at NationalSite.

Implement two functions:

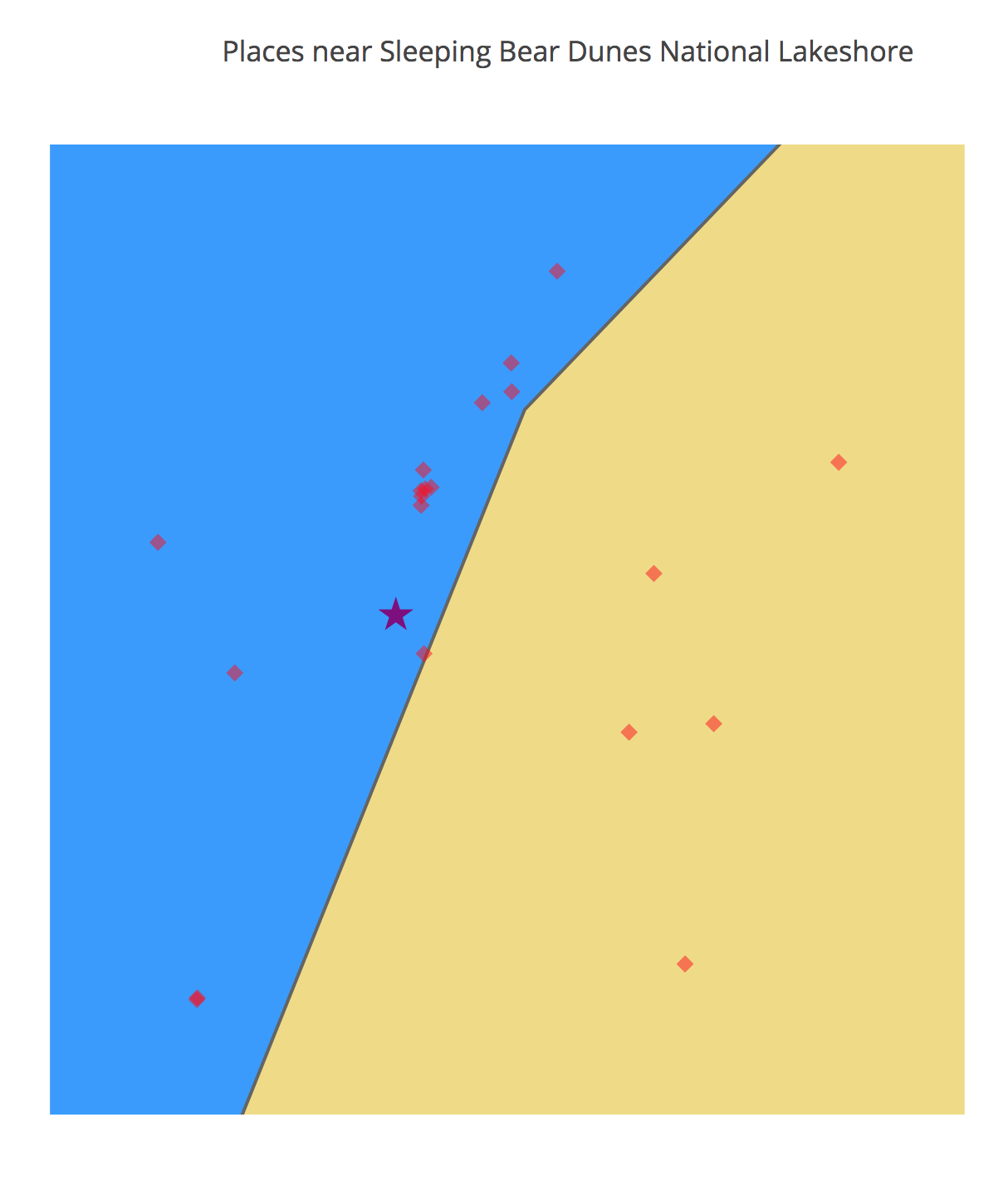
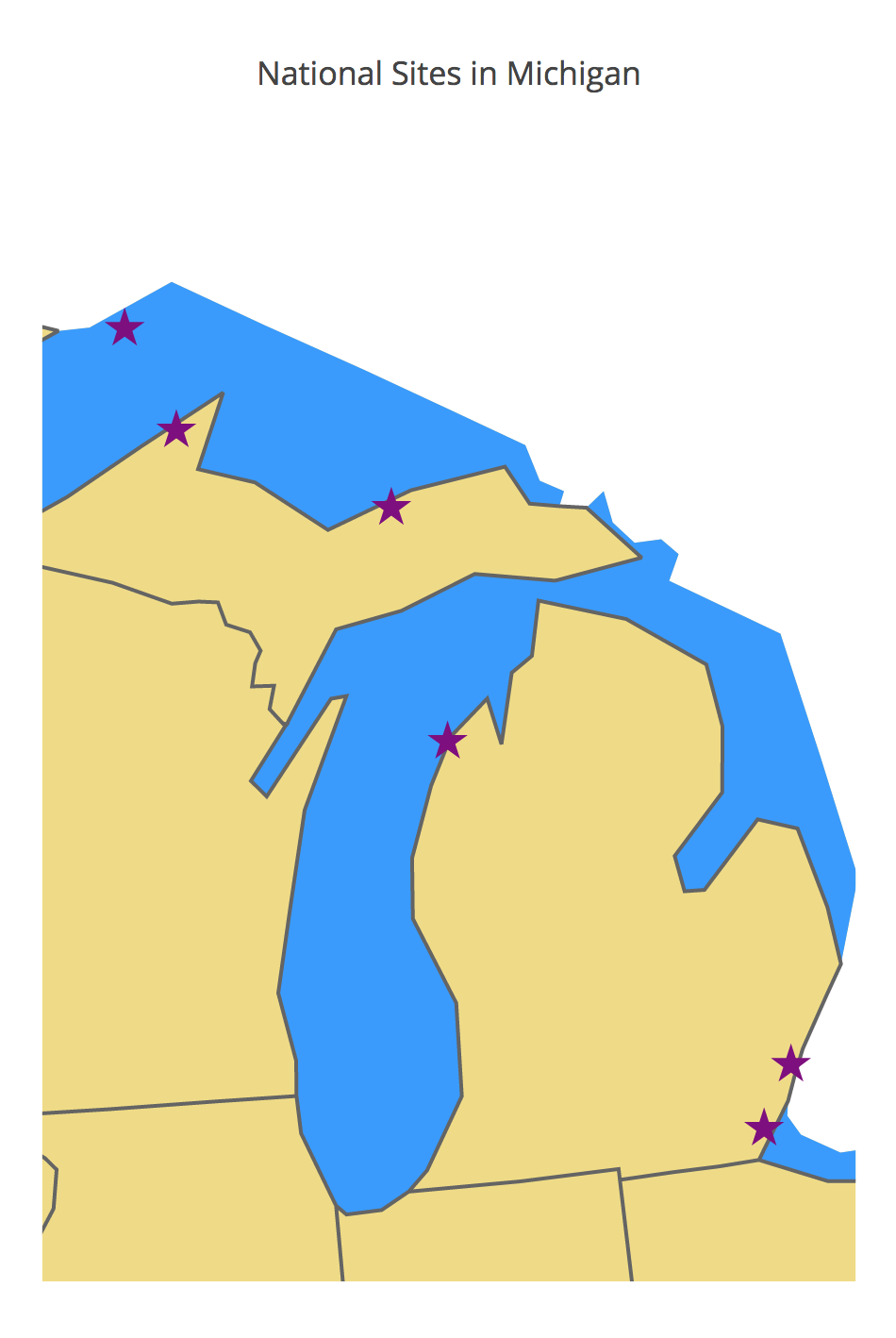
plot\_sites\_for\_state(state\_abbr) and plot\_nearby\_for\_site(site\_object)

Here are some details about each function:

* plot\_sites\_for\_state(state\_abbr):
  + Takes a state abbreviation
  + Creates a plotly map scatter plot (or mbox scatter plot) that contains all of the NationalSites found for that state *that Google Places was able to find GPS coordinates for*.
    - Any Sites that don’t have GPS coordinates should be removed before creating the map
  + The map should be centered and scaled appropriately so that all of the sites are visible and that there is a reasonable amount of “padding” around the edges of the map (i.e., so that all of the sites are comfortably within the map frame and not all the way at the edge)
  + All Sites should be displayed with the same type of marker, and each should display the name of the site when a user hovers over the marker (this is the default behavior in plotly if each data point has a ‘text’ field correctly set).
* plot\_nearby\_for\_site(site\_object)
  + Takes a NationalSite object
  + Creates a plotly map scatter plot (or mbox scatter plot) that contains all of the NearbyPlaces for the specified site.
    - If a NationalSite is provided that Google Places can’t find GPS coordinates for, the map should not be created (you can handle the error however you deem appropriate—but your program should not crash)
  + The map should be centered and scaled appropriately so that all of the places are visible and that there is a reasonable amount of “padding” around the edges of the map (i.e., so that all of the sites are comfortably within the map frame and not all the way at the edge)
  + The NationalSite should be displayed with a *different* marker than the NearbyPlaces. Note that the NationalSite may be returned as a result by Google Places, in which case it needs to be removed before the map is plotted. The NationalSite and all NearbyPlaces should display their name when a user hovers over the marker in plotly.

Here are examples of each:  
On the left is the result of calling plot\_sites\_for\_state(‘mi’).

On the right is the result of calling plot\_nearby\_for\_site(NationalSite(‘National Lakeshore’, ‘Sleeping Bear Dunes’))



Don’t worry about the fact that some of the markers appear to be off by a few fractions of a degree. This seems to have something to do with the projection we are using for plotly in our code (‘albers usa’) which doesn’t agree with the coordinates being produced by Google. If the data is correct and the maps are more or less in the right area, you will not get points off.

Grading:

* [20 points] Correct implementation of plot\_sites\_for\_state(state\_abbr)
* [20 points] Correct implementation of plot\_nearby\_for\_site(site\_object)

# Part 4 (40 Points)

Make the program interactive. Here is a list of commands your program should accept and how it should handle them:

list <stateabbr>

available anytime

lists all National Sites in a state

valid inputs: a two-letter state abbreviation

nearby <result\_number>

available only if there is an active result set

lists all Places near a given result

valid input for <result number>: an integer 1 to len(result\_set\_size)

map

available only if there is an active result set

displays the current results on a map

exit

exits the program

help

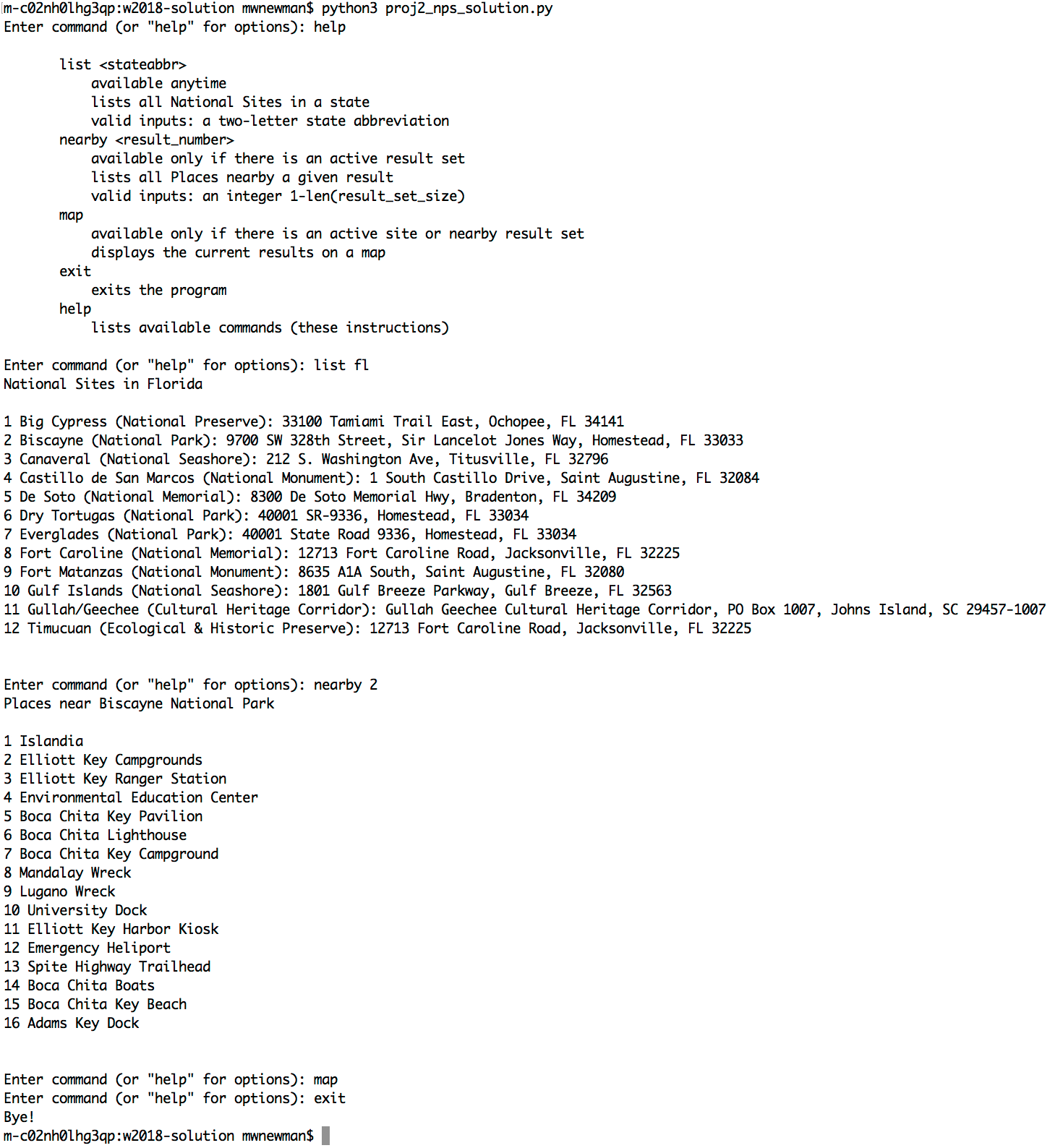
lists available commands (these instructions)

Note: a “result set” here refers to a list of NationalSites for a state or a list of NearbyPlaces for a NationalSite. You can implement this concept however you like, as long as the above semantics are preserved. **This part shouldn’t be run when running the test, but it needs to be run when running the proj2\_nps.py file.**

A few notes on user experience:

* When the program starts, it is nice to tell the user what she is able to do. Don’t just give the user a blinking cursor with no information about what input she can provide. (This may seem obvious, but you’d be surprised how many such programs we have received.)
* Please number your result lists, so the user can see what number she is supposed to enter to search for a nearby place. Otherwise, the user has to count, which is…not great user experience.
* Your program will be much more usable if at each point the user knows what her options are. So, if the user searches first types “list mi” and your program provides her with a list of national sites in Michigan, you might want to then print above the user input prompt something like ‘Type “nearby <result number>” to search for places near one of the national sites above, “map” to map the list of national sites, or ‘list <state>” to do a search for another state:’ Or a less verbose version of something like that. Even though we will know what to do, it’s a good practice to write your programs so that they could be used by someone who is not familiar with it.

Here is a sample run of the program (albeit, with less elegant user experience than what is described above):



Grading:

* [8 points] Implement ‘list’ command correctly
* [8 points] Implement ‘nearby’ command
* [8 points] Implement ‘map’ command
* [4 points] Implement ‘help’ command
* [4 points] Implement ‘exit’ command
* [8 points] Handle bad inputs elegantly