CS1210 Spring 2020 DS10
Topic: caching

- A very useful tool in many programming situations, and especially in programs that access web services
- Basic idea: save results of things you've already computed (especially things that take a while to computer or retrieve from the internet) so that if you need them again later, you don't have to re-compute
- One example:
  - Earlier this semester, demonstrated basic recursive code to Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, …
  - Works okay on most people's computers up to high 30s but takes too long on say fib(50)
  - Why is it so slow? Basic process is simple – adding two smaller values. But the particular implementation unnecessarily recomputes values *many* times and ends up being enormously inefficient. For example, to compute fib(8), the code first compute fib(6) and then it computes fib(7). But to compute fib(7) it completely recomputes fib(6)! We can easily avoid this.
  - See fib and fib2 in fibCache.py

Example of using caching in web applications: lowering the number of calls to Google's geocoding service

- Many web APIs, especially when free/basic versions have limits on number of queries you can make before paying, and often also limit the rate at which you can make queries.
  - E.g. 600 queries per day and/or 15 queries per second.
  - certainly can be used in HW10 but is not required.
- The problem: Google's geocoding service, used by the geocodeAddress function of HW8 Q2 and HW9, limits both the number and rate at which you can make queries. We probably won't exceed those quotas but using caching in geocodeAddress provides a good example (and likely will make some parts of your program faster). You are not *required* to add caching to your HW8 or 9 code but you may if you wish.

1. Add your API key (or Prof. Cremer’s) on line 10 to geocodeonly.py. Unlike for HW8 Q2 and HW9, you may use Prof. Cremer’s API key for DS 10 without losing a point.
2. Load geocodeonly.py into Python. The file contains
   – geocodeAddress
   – testGeocode, a function that takes an integer, numLoops, as argument (default value 1) and makes 10*numLoops geocodeAddress calls.
3. Run testGeocode(). You will see some lat/lng pairs for the ten city locations queried in testGeocode.
4. testGeocode takes a number of iterations as input (1 is the default). If we gave a larger number as input – e.g. testGeocode(100) - and if the Python calls to the Google API were fast enough, we might eventually exceed our quota or the API rate limit (unless we upgraded our account and paid Google more.)

Using caching in geocodeAddress

IDEA: because our HW8 and 9 programs might make many geocode queries involving the same address, let’s **save results** from Google geocoding service locally, so we don’t waste quota asking Google things we’ve already asked. Where should we save geocode information?

- Use a dictionary (call it geoDict) where each entry has an address as key and a lat/lng tuple as value. E.g.

  {'iowa city': (41.6611277, -91.5301683), 'new york': (40.7127837, -74.0059413)}

- We’ll then modify geocodeAddress in two ways:
  - Before making a request to Google’s service, check if we have an entry for the address in geoDict. E.g. check geoDict[‘Iowa City’].
  - If so, return it! (don’t bother Google)
  - If not, ask Google. But now, before returning the result, save it in geoDict so that next time we want to know about this address we won’t ask Google again.

Code implementing the use of the cache

- add top-level line that creates and initializes a global variable

```
geoDict = {}
```

- add to geocodeAddress
  - at very beginning:

```
if addressString in geoDict:
    return geoDict[addressString]
```

  - at end, just before return line:

```
geoDict[addressString] = result
```

Now load file into Python and test!

Example test transcript

```
>>> geoDict
{}
>>> geocodeAddress('iowa city')
(41.6611277, -91.5301683)
>>> geoDict
{'iowa city': (41.6611277, -91.5301683)}
>>> geocodeAddress('new york')
(40.7127837, -74.0059413)
>>> geoDict
{'iowa city': (41.6611277, -91.5301683), 'new york':
(40.7127837, -74.0059413)}
```

You can see your little local geocode information ‘cache’ growing – you won’t need to ask Google for Iowa City or New York’s lat/lng again!

Try testGeocode again

Now try test testGeocode() function again:

1) run testGeocode(1). You should see city names and associated lat/lng pairs printing fairly slowly, one or two per second perhaps.
2) ”look” at geoDict in the shell

   ```
   >>> geoDict
   ```

   You should see a dictionary full of location : lat/lng pairs
3) run testGeocode(5). The cities and lat/lng pairs should print very quickly this time. geocodeAddress never needs to actually make a request to Google.

After the first geocode request for each location, all the rest are repeats so we don’t ask Google again; we just get the answer directly from geoDict

BUT WAIT! What if we stop/close Python? We'd need to rebuild the dictionary next time we run the program. Can we easily save the dictionary in a file and reload it next time we want to run the program? YES!

So far, we can cache geocode query results in a dictionary and use them to avoid calls to Google's service. But we want to keep the results "permanently." How can we save the dictionary to a file before quitting Python?

- Python provides JSON tools that convert dictionaries to JSON, which we can then easily save to a file and also easily read back in!

- Copy the two functions readGeoDict and saveGeoDcit from readSaveGeoDict.py into your geocodeonly.py file (yes, it would work to import the file but do NOT do so for DS10. Copy the functions into your main file and submit just that one file with all of the code.)

TEST AGAIN!

Test saving/reading geoDict

1. load geocodeonly.py (with the 2 new functions)
2. Execute testGeocode() in the Python interpreter
3. Execute saveGeoDict()
4. Close Python
5. Look in whichever folder you've been working in. You should see a file called geodict.json
6. View it with a text editor. It should look like a Python dictionary.
7. Load geocodeonly.py into a new Python.
8. Execute readGeoDict()
9. Type geoDict at the Python prompt. You should see that geoDict is a dictionary filled with the data from the saved file. (And if you try testGeocode() now, it will execute quickly because it doesn't have to talk at all to Google.)

Submit the revised geocodeonly.py to the DS10 assignment on ICON

Optional (do later): add geocode caching to HW8 Q2 and/or HW9

1. add call to readGeoDict when you initialize HW8 Q2 or HW9 execution. E.g. add readGeoDict() somewhere in your initialization code.
2. replace original geocodeAddress with today's new one
3. Add a line after initializing tkinter:

```
rootWindow = tkinter.Tk()
rootWindow.protocol("WM_DELETE_WINDOW", handleCloseRootWindow)
```

4. Add function that calls saveGeoDict when tkinter main window is closed.

```
def handleCloseRootWindow():
  rootWindow.destroy()
  saveGeoDict()
```