README:
Data-gathering Utilities for Job Search Websites
The Kovani Group
Michael Smith
9/19/15

Table of Contents

1. Basic Usage

1.1. Get Postings

a) Implemented in this Python package are scrapers for job search websites. Scrapers inherit from the same abstract PostingScraper class, with a "get_postings" method. Calling this method on any scaper object with a query and the number of pages desired yields a search on the specified site, and the scraper gathers at most the specified number of pages of results.

1.2. Specifying Location

a) When appropriate, scrapers have a "base url" (e.g. Craigslist) or a "location" (e.g. Indeed) parameter given when instantiating them.
   - Locations restrict the results by searching the query in get_postings to the given location. Note that this location must be a valid location as per the website in question - it's passed to the "location" parameter of the search query.
   - Base urls perform the same function, but in a different way that conforms to a website's framework.
     - Craigslist for example makes a user choose his base url before searching, so to search in Baltimore a user would have to first navigate to baltimore.craigslist.org
     - Visiting http://www.craigslist.org/about/sites yields a list of base urls for Craigslist.

2. Example Use

2.1. Get Postings

a) The "get_postings" function returns a list of Posting objects. The function's documentation is reproduced below:

```
def get_postings(self, query, pages=1):
        """
        Gather results, given a query and the number of pages of results desired.
        A posting is a dict with the following minimum attributes:
            - text description
            - source
            - unique id
            - title
            - date posted
            - url
        Postings from websites may contain more information, e.g., price or
location
        :param query: query to use on the site
        :param pages: number of pages of results desired
        :return: a list of postings, each of which is a dict.
        """
```

2.2. Posting Class

a) Postings are custom dictionaries that override the __hash__ and __eq__ methods based on fuzzy text matching of the descriptions of the postings, as well as comparing the sources of the postings. One can call "get_data_backing" on a posting to retrieve its backing dictionary, if desired. Things to note:
   - Date_posted are datetime objects
   - All other data fields are strings

2.3. Error handling

a) Individual Data-gathering Errors, Silently Handle
   - As written, the scrapers gracefully ignore appropriate errors when attempting to get individual pieces of information from websites. Postings in these instances will simply be lacking the individual data that it could not obtain. This would indicate that a website's structure could have undergone possibly significant changes, such that the data-gathering code will need to be revised. There will not be a ValueError listed in the standard error stream.

b) Ignoring External Redirections, Notification
   - Scrapers will ignore postings whose links redirect to a website that is external from their own base website. This is an extension of the aforementioned 'individual errors', as all the gathering would fail on an object if the page is not as expected (which would happen if a link redirected to a page in an unknown format, like one external to the website in question). There will be a ValueError in the standard error stream.

c) Larger Errors, Notification
   - Scrapers simply return zero postings if a more catastrophic error occurs, e.g. a

website is unreachable. There will be a ValueError in the standard error stream.

3. Code example

```
from artichoke_clean import CraigslistScraper
from artichoke_clean import UpworkScraper
from artichoke_clean import GlassdoorScraper
from artichoke_clean import IndeedScraper
from artichoke_clean import SimplyhiredScraper
from artichoke_clean import ZiprecruiterScraper

scrapers = list()
scrapers.append(CraigslistScraper(base_url="http://baltimore.craigslist.org"))
scrapers.append(UpworkScraper())
scrapers.append(GuruScraper())
scrapers.append(IndeedScraper(location="baltimore"))
scrapers.append(SimplyhiredScraper(location="Baltimore, MD"))
scrapers.append(ZipRecruiterScraper(location="baltimore"))
result_data = []  # list of dicts
for scraper in scrapers:
    for p in scraper.get_postings("computer thing", pages=50):
        for k, v in p.items():  # support iteration, and all the standard dict
calls
            print k, " => ", v
      result_data.append(p.get_data_backing())
```