freestuffs Documentation

Release 0.1

Fenimore Love

October 19, 2016

Contents

This is a Python 3.x package which scrapes free stuff from Craigslist. freestuffs is under the MIT license. Hosted on Github.

This package can be used to create a web application, such as the Treasure-map (source), or for use on Twitter.

CHAPTER 1

Installation

Install using pip, freestuffs requires Python 3 and these dependences:

- requests
- geopy
- folium
- BeautifulSoup4
- unidecode

Install:

```
pip install freestuffs
```

Documentation:

1.1 Introduction

1.1.1 Free Stuffs!

This is a Python 3.x package which scrapes free stuff from Craigslist. freestuffs is under the MIT license. Check out the source code and the docs.

- Using StuffScraper one can gather a list of free stuffs.
- Using StuffCharter, one can create an HTML map of the free stuffs.

This package can be used to create a web application, such as the Treasure-map (source), or for use on Twitter.

1.1.2 Installation

Install using pip, requires Python 3 and these dependences:

- requests
- geopy
- folium
- BeautifulSoup4
- unidecode

Install: .. code-block:: bash

    pip install freestuffs

1.1.3 Getting Started

Stuffs

The stuff class corresponds to a Craiglist free stuff posting. It's basic characteristics include title and location. Notably, there is no price attribute. If the posting has no image, the Wikipedia no-image image is used in it's place.

```
>>> from freestuffs.stuff_scraper import StuffScraper
>>> stuffs = StuffScraper('montreal', 5).stuffs
>>> print(stuffs[0])
what: free shelves
 where: Workman St, montreal
 link: http://montreal.craigslist.ca/zip/5629811181.html
 image: https://images.craigslist.org/00r0r_4p06sM5Hn4O_300x300.jpg
```

Scape Stuffs

The StuffScraper class will scrape Craiglist for free stuff.

```
>>> from freestuffs.stuff_scraper import StuffScraper
>>> stuffs = StuffScraper('montreal', 5).stuffs # precise=False
>>> print(stuffs[0].thing)  # Title
Meubles / furniture
```

In order for the scraper to automatically scrape for latitude and longitude coordinates, pass in the parameter precise=True into the constructor.

```
>>> from freestuffs.stuff_scraper import StuffScraper
>>> stuffs = StuffScraper('montreal', 5, precise=True).stuffs
>>> print(stuffs[0].coordinates)
['45.617854', '-73.633931']
```

Chart Stuffs

The StuffCharter class will produce a folium Map object populated with free stuff from the StuffScraper.

```
>>> from freestuffs.stuff_scraper import StuffScraper
>>> from freestuffs.stuff_charter import StuffCharter
>>> stuffs = StuffScraper('montreal', 5, precise=True).stuffs
>>> stuffs_chart = StuffCharter(stuffs)
call save_map(path) generate html map
>>> type(map.treasure_map)
<class 'folium.folium.Map'>
```

The StuffCharter object is a wrapper around the folium.Map. Call `save_map(HTML_PATH, CSS_PATH)`

```
>>> stuffs_chart.save_map('webmap', 'static/style.css')
```

This function creates a directory if it is not found in the path. Call instead `save_test_map()` to generate an HTML map in the current directory.

Legend

- The smaller the posting, the older it is.
- The darker the border, the higher the amount of overlap.

Triage

The triage checks for regex search in this order:

1. Red are furniture "(wood, shelf, shelves, table, chair, scrap, desk)".
2. Blue are electronics "(tv, sony, ecran, speakers, wire, electronic, saw, headphones, arduino)".
3. Black are the "desired" stuffs "(book, games, cool, guide, box)".
4. White is default (no regex search matches).

1.1.4 Support

The easiest way to get support is to open an issue on **Github_**.

1.2 Cookbook

1.2.1 Stuffs

The stuff class corresponds to a Craiglist free stuff posting. It's basic characteristics include title and location. Notably, there is no price attribute. If the posting has no image, the Wikipedia no-image image is used in it's place.

```
>>> from freestuffs.stuff_scraper import StuffScraper
>>> stuffs = StuffScraper('montreal', 5).stuffs
>>> print(stuffs[0])
what: free shelves
 where: Workman St, montreal
 link: http://montreal.craigslist.ca/zip/5629811181.html
 image: https://images.craigslist.org/00r0r_4p06sM5Hn4O_300x300.jpg
```

1.2.2 Scape Stuffs

The StuffScraper class will scrape Craiglist for free stuff. The two required args are the city name and the quantity of stuff to scrape. The city name **must** conform to the Craiglist url name. Cities like New York, are then 'newyork'.

```
>>> from freestuffs.stuff_scraper import StuffScraper
>>> stuffs = StuffScraper('montreal', 5).stuffs # precise=False
>>> print(stuffs[0].thing)  # Title
Meubles / furniture
```

In order for the scraper to automatically scrape for latitude and longitude coordinates, pass in the parameter precise=True into the constructor.

```
>>> from freestuffs.stuff_scraper import StuffScraper
>>> stuffs = StuffScraper('montreal', 5, precise=True).stuffs
>>> print(stuffs[0].coordinates)
['45.617854', '-73.633931']
```

Otherwise, one can call `stuffs[0].find_coordinates()` in order to set (and scrape) the stuff coordinates one by one.

Pass in `use_cl=true` in order to ask for user input and override the location entered in the __init__.:

```
>>> from freestuffs.stuff_scraper import StuffScraper
>>> stuffs = StuffScraper('ill decide later', 1, use_cl=True).stuffs
What major city are you near? (or, 'help') newyork
>>> print(stuffs[0].location)
East Harle, New York
```

1.2.3 Chart Stuffs

The StuffCharter class will produce a folium Map object populated with free stuff from the StuffScraper. The StuffCharter object is a wrapper around the folium.Map.

```
>>> from freestuffs.stuff_scraper import StuffScraper
>>> from freestuffs.stuff_charter import StuffCharter
>>> stuffs = StuffScraper('montreal', 5, precise=True).stuffs
>>> stuffs_chart = StuffCharter(stuffs)
call save_map(path) to generate html map
>>> type(map.treasure_map)
<class 'folium.folium.Map'>
```

Call `save_map(HTML_PATH, CSS_PATH)` in order to save an HTML map from the folium.Map object. (equivelant to calling `folium.Map.save(path)`)

```
>>> stuffs_chart.save_map('webmap', 'static/style.css')
```

This function creates a directory if it is not found in the path. Call instead `save_test_map()` to generate an HTML map in the current directory.

Optionally pass in an `address` or `zoom` level into its construction. Otherwise if `do_create_map` is `False`, these attributes can be modified manually.

```
>>> from freestuffs.stuff_scraper import StuffScraper
>>> from freestuffs.stuff_charter import StuffCharter
>>> stuffs = StuffScraper('montreal', 5, precise=True).stuffs
>>> stuffs_chart = StuffCharter(stuffs, zoom=15, do_create_map=False)
>>> stuffs_chart.zoom = 10 # default 13
>>> stuffs_chart.create_map()
call save_map(path) to generate html map
```

The stuff markers are colored circles in diminishing order; the small the circle, the older the posting (this prevents inaccessible overlaps).

And you can add an address (not zoom) after the map has been created:

```
>>> stuffs_chart.add_address('5989 Rue du Parc, Montreal, Quebec')
>>> print(stuffs_chart.address)
5989 Rue du Parc, Montreal, Quebec
```

And why stop at one address maker(the `address` attribute will always be the last address added):

```
>>> stuffs_chart.add_address('5989 Rue du Parc, Montreal, Quebec')
>>> stuffs_chart.add_address('604 Rue Saint Joseph, Montreal, Quebec')
>>> print(stuffs_chart.address)
604 Rue Saint Joseph, Montreal, Quebec
```

Override the css by adding links to the folium object header:

```
>>> import folium
>>> osm_map = stuffs_chart.treasure_map
>>> folium_figure = osm_map.get_root()
>>> folium_figure.header._children['bootstrap'] = folium.element.CssLink('/static/css/style.css')
```

To use the treasure_map as a template in a python web app, the leaflet bootstrap css might conflict with the user defined styles. Before saving the map, add a CssLink.

The fastest way to get a map up and running, is to pass `is_testing=True` into the constructor:

```
>>> from freestuffs.stuff_scraper import StuffScraper
>>> from freestuffs.stuff_charter import StuffCharter
>>> stuffs = StuffScraper('montreal', 5, precise=True).stuffs
>>> stuffs_chart = StuffCharter(stuffs, is_testing=True)
BEWARNED, this map is likely inaccurate:
Craigslist denizens care not for computer-precision
```

1.3 freestuffs package

1.3.1 Submodules

1.3.2 freestuffs.city_list module

Craiglist friendly city names.

Run this script inorder to print out a dict of Craiglist city names, for use in creating a valid url.

Attributes:

- **CITIES – A dict of cities, with the human-friendly** name as the key and url friendly name as the value.

`freestuffs.city_list.`**`scrape_cities`**`()`
Scrape city and link for dict of valid names.

1.3.3 freestuffs.stuff module

This houses the Stuff class.

Use stuff_scraper in order to gather a list of stuffs. For testing, the reverse Geolocator is at nominatim.openstreetmap.org.

Example usage:

```
>>> from stuff_scraper import StuffScraper
>>> stuffs = StuffScraper('montreal', 5).stuffs
>>> print(stuffs[0].thing)  # Title
Meubles / furniture
>>> stuffs[0].find_coordinates() # pass precise=True in constructor
>>> print(stuffs[0].coordinates) # to automatically fetch coordinates
['45.617854', '-73.633931']
```

class `freestuffs.stuff.`**`Stuff`**(*thing*, *url*, *location*, *image*, *user_location*)
Bases: `object`

A freestuff Craigslist object.

Fill this object with the information from a craigslist page. (There is no price attribute, because it is designed for invaluable things). The precise coordinates are not initially set, because they require significantly more requests. See class method get_coordinates().

Attributes:

- thing – title of object passed explicitly
- url – constructed from url, implicit
- image – passed explicitly
- user_location – passed explicitly, requires clean up
- coordinates – array of longitude and latitude

Keyword arguements:

- thing –
- url –
- location –
- image –
- user_location – must conform to valid craiglist url

find_coordinates()
Get and set longitude and Latitude

Scrape individual posting page, if no coordinates are found, cascade precision (try location, try user_location, or set to zero). Returns an array, first latitude and then longitude.

1.3.4 freestuffs.stuff_charter module

Chart where free things are.

The StuffCharter class is a wrapper around the folium openstreetmap python object, which in turn generates a leaflet map.

Example usage:

```
>>> from stuff_scraper import StuffScraper
>>> from stuff_charter import StuffCharter
>>> stuffs = StuffScraper('montreal', 5, precise=True).stuffs
>>> treasure_map = StuffCharter(stuffs)
call save_map(path) generate html map
>>> treasure_map.save_test_map() # saves map in current dir
BEWARNED, this map is likely inaccurate:
Craigslist denizens care not for computer-precision
```

class freestuffs.stuff_charter.**StuffCharter**(*stuffs*, *address=None*, *zoom=13*, *do_create_map=True*, *is_testing=False*, *is_flask=False*)

Bases: object

Post folium map of freestuffs.

After constructing Mappify map object, call create_map and pass in map_path in order to create the HTML map.

Attributes:

- treasure_map – an OSM folium map object
- stuffs – list of free stuff
- user_location – the user's location
- start_coordinates – the origin coordinates for city
- zoom – default map zoom

Keyword arguments:

- stuffs – a list of stuff objects
- address – for an optional map marker of the user address.
- **do_create_map – set to False to override modify attributes** before create_map.
- is_testing – use to test module from commandline
- is_flask – automatically create map for treasure-map
- zoom – the map default zoom level

add_address(*_address*)
Add address to folium map

create_map(*is_testing=False*, *is_flask=False*)
Create a folium Map object, treasure_map.

treasure_map can be used to save an html leaflet map. This method is called automatically on __init__ unless do_create_map is set to False.

Keyword arguments:

- is_testing – creates a map in webmap directory
- is_flask – creates a flask map

find_city_center(*location*)
Return city center longitude latitude.

save_flask_map()
Create html map in flask server.

save_map(*map_path*, *css_path=None*)
Create html map in map_path.

Keyword arguments:

- map_path – the path to create_map in
- **css_path – the path to override css** (defaults to bootstrap via folium)

save_test_map()
Create html map in current directory.

Should have python -m http.server running in directory

sort_stuff(*stuff*)
Return a color according to regex search.

1.Furniture pattern, red

2.Electronics pattern, blue

3.Miscellaneous pattern, black

4.no match, white

sort_stuff will return with the first pattern found in that order.

TODO:

- Set and patterns as modifiable attributes.

1.3.5 freestuffs.stuff_scraper module

This module is a Craigslist scraper.

Example usage:

```
>>> from stuff_scraper import StuffScraper
>>> stuffs = StuffScraper('montreal', 5).stuffs
>>> print(stuffs[0].thing) # Print title
Meubles / furniture
```

class `freestuffs.stuff_scraper.`**`StuffScraper`**(*place*, *_quantity*, *precise=False*, *use_cl=False*)

Bases: `object`

The freestuffs Craigslist scraper.

Compile parrellel lists of stuff attributes in order to store a freestuffs list, with an option for including stuff coordinates.

Attributes:

- stuffs – a list of stuff objects
- soup – bs4 soup of Craiglist page
- place – the city to search, in Craigslist friendly format
- locs, things, images, urls – stuff attributes lists
- quantity – how many stuffs gathered

Keyword arguments:

- _quantity – how many stuffs to gather
- **precise – A boolean to explicitly use geolocator** and crawl individual posting URL
- use_cl – user input for place

`get_images`(*_soup*)

Scrape images.

Uses wikpedia No-image image if no image is found.

Keyword arguments:

- soup - bs4 object of a Craiglist freestuffs page

`get_locations`(*user_place*, *_soup*)

Scape locations.

Returns a list of locations, more or less precise. Concatnate user_place to string in order to aid geolocator in case of duplicate location names in world. Yikes.

Keyword arguments:

- user_place – the city, in Craigslist format
- soup – bs4 object of a Craiglist freestuffs page

get_things(*_soup*)
Scrape titles.

Keyword arguments: - soup - bs4 object of a Craiglist freestuffs page

get_urls(*_soup*)
Scrape stuff urls.

Keyword arguments:

- soup - bs4 object of a Craiglist freestuffs page

refine_city_name(*user_place*)
Refine location for two-word cities.

setup_place()
Take cl input of user location.

1.3.6 freestuffs.treasure_script module

1.3.7 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

f

A

C

F

G

R

S