

---

**geograpy3**

**Sep 21, 2020**



---

## Contents:

---

<b>1</b>	<b>geograpy package</b>	<b>1</b>
1.1	Submodules	1
1.2	geograpy.extraction module	1
1.3	geograpy.labels module	1
1.4	geograpy.locator module	2
1.5	geograpy.places module	5
1.6	geograpy.prefixtree module	5
1.7	geograpy.utils module	6
1.8	Module contents	6
<b>2</b>	<b>setup module</b>	<b>9</b>
<b>3</b>	<b>tests package</b>	<b>11</b>
3.1	Submodules	11
3.2	tests.test_extractor module	11
3.3	tests.test_locator module	12
3.4	tests.test_places module	13
3.5	tests.test_prefixtree module	13
3.6	Module contents	13
<b>4</b>	<b>Indices and tables</b>	<b>15</b>
	<b>Python Module Index</b>	<b>17</b>
	<b>Index</b>	<b>19</b>



### 1.1 Submodules

### 1.2 geograpy.extraction module

**class** `geograpy.extraction.Extractor` (*text=None, url=None, debug=False*)

Bases: `object`

Extract geo context for text or from url

**find\_entities** (*labels=['GPE', 'GSP', 'PERSON', 'ORGANIZATION']*)

Find entities with the given labels set `self.places` and returns it :param labels: Labels: The labels to filter

**Returns** List of places

**Return type** list

**find\_geoEntities** ()

Find geographic entities

**Returns** List of places

**Return type** list

**set\_text** ()

Setter for text

**split** ()

simpler regular expression splitter with not entity check

hat tip: <https://stackoverflow.com/a/1059601/1497139>

### 1.3 geograpy.labels module

Created on 2020-09-10

@author: wf

```
class geograpy.labels.Labels
    Bases: object

    NLTK labels

    default = ['GPE', 'GSP', 'PERSON', 'ORGANIZATION']

    geo = ['GPE', 'GSP']
```

## 1.4 geograpy.locator module

The locator module allows to get detailed city information including the region and country of a city from a location string.

Examples for location strings are:

Amsterdam, Netherlands Vienna, Austria Vienna, IL Paris - Texas Paris TX

the locator will lookup the cities and try to disambiguate the result based on the country or region information found.

The results in string representationa are:

Amsterdam (NH(North Holland) - NL(Netherlands)) Vienna (9(Vienna) - AT(Austria)) Vienna (IL(Illinois) - US(United States)) Paris (TX(Texas) - US(United States)) Paris (TX(Texas) - US(United States))

Each city returned has a city.region and city.country attribute with the details of the city.

Created on 2020-09-18

@author: wf

```
class geograpy.locator.City
    Bases: object

    a single city as an object

    static fromGeoLite2 (record)

class geograpy.locator.Country
    Bases: object

    a country

    static fromGeoLite2 (record)
        create a country from a geolite2 record

    static fromPyCountry (pcountry)

        Parameters pcountry (PyCountry) – a country as gotten from pycountry

        Returns the country

        Return type Country

class geograpy.locator.Locator (db_file=None, correctMisspelling=False, debug=False)
    Bases: object

    location handling

    cities_for_name (city_name)
        find cities with the given city_name
```

**Parameters** `city_name` (*string*) – the potential name of a city

**Returns** a list of city records

**correct\_country\_misspelling** (*name*)

correct potential misspellings :param name: the name of the country potentially misspelled :type name: string

**Returns** correct name of unchanged

**Return type** string

**db\_has\_data** ()

check whether the database has data / is populated

**Returns** True if the cities table exists and has more than one record

**Return type** boolean

**disambiguate** (*country, regions, cities*)

try determining country, regions and city from the potential choices

**Parameters**

- **country** (*Country*) – a matching country found
- **regions** (*list*) – a list of matching Regions found
- **cities** (*list*) – a list of matching cities found

**Returns** the found city or None

**Return type** *City*

**getCountry** (*name*)

get the country for the given name :param name: the name of the country to lookup :type name: string

**Returns** the country if one was found or None if not

**Return type** country

**getGeolite2Cities** ()

get the Geolite2 City-Locations as a list of Dicts

**Returns** a list of Geolite2 City-Locator dicts

**Return type** list

**static getInstance** (*correctMisspelling=False, debug=False*)

get the singleton instance of the Locator. If parameters are changed on further calls the initial parameters will still be in effect since the original instance will be returned!

**Parameters**

- **correctMisspelling** (*bool*) – if True correct typical misspellings
- **debug** (*bool*) – if True show debug information

**isAmbiguousPrefix** (*name*)

check if the given name is an ambiguous prefix

**Parameters** **name** (*string*) – the city name to check

**Returns** True if this is a known prefix that is ambiguous that is there is also a city with such a name

**Return type** bool

**isISO** (*s*)

check if the given string is an ISO code

**Returns** True if the string is an ISO Code

**Return type** bool

**isPrefix** (*name, level*)

check if the given name is a city prefix at the given level

**Parameters**

- **name** (*string*) – the city name to check
- **level** (*int*) – the level on which to check (number of words)

**Returns** True if this is a known prefix of multiple cities e.g. “San”, “New”, “Los”

**Return type** bool

**is\_a\_country** (*name*)

check if the given string name is a country

**Parameters** **name** (*string*) – the string to check

**Returns** if pycountry thinks the string is a country

**Return type** True

**locate** (*places*)

locate a city, region country combination based on the places information

**Parameters** **places** (*list*) – a list of place tokens e.g. “Vienna, Austria”

**Returns** a city with country and region details

**Return type** *City*

**locator** = None

**places\_by\_name** (*place\_name, column\_name*)

get places by name and column :param place\_name: the name of the place :type place\_name: string :param column\_name: the column to look at :type column\_name: string

**populate\_Cities** (*sqlDB*)

populate the given sqlDB with the Geolite2 Cities

**Parameters** **sqlDB** (*SQLDB*) – the SQL database to use

**populate\_PrefixAmbiguities** (*sqlDB*)

create a table with ambiguous prefixes

**Parameters** **sqlDB** (*SQLDB*) – the SQL database to use

**populate\_PrefixTree** (*sqlDB*)

calculate the PrefixTree info

**Parameters** **sqlDB** – the SQL Database to use

**Returns** the prefix tree

**Return type** *PrefixTree*

**populate\_db** (*force=False*)

populate the cities SQL database which caches the information from the GeoLite2-City-Locations.csv file

**regions\_for\_name** (*region\_name*)

get the regions for the given region\_name (which might be an ISO code)

**Parameters** `region_name` (*string*) – region name

**Returns** the list of cities for this region

**Return type** list

**class** `geograpy.locator.Region`

Bases: `object`

a Region (Subdivision)

**static fromGeoLite2** (*record*)

create a region from a Geolite2 record

**Parameters** `record` (*dict*) – the records as returned from a Query

**Returns** the corresponding region information

**Return type** *Region*

## 1.5 geograpy.places module

**class** `geograpy.places.PlaceContext` (*place\_names, setAll=True*)

Bases: `geograpy.locator.Locator`

Adds context information to a place name

**get\_region\_names** (*country\_name*)

**setAll** ()

Set all context information

**set\_cities** ()

set the cities information

**set\_countries** ()

get the country information from my places

**set\_other** ()

**set\_regions** ()

## 1.6 geograpy.prefixtree module

Created on 2020-09-20

@author: wf

**class** `geograpy.prefixtree.PrefixTree`

Bases: `object`

prefix analysis and search

see <http://p-nand-q.com/python/data-types/general/tries.html>

**add** (*name*)

add the given name to the prefix Tree

**Parameters** `name` (*string*) – the name to add

**add2Table** (*prefix, prefixStr, table, level*)

recursively add prefix tree entries to a table

**Parameters**

- **prefix** (*dict*) – the dictionary to start with
- **prefixStr** (*string*) – the prefix string up to this level
- **table** (*list*) – a “flat” list of dicts as a table
- **level** (*int*) – the level (length of word sequence) on which to add

**countStartsWith** (*namePrefix*)

count how many entries start with the given namePrefix

**Parameters** **namePrefix** (*string*) – the prefix to check

**getCount** ()

get my total count

**Returns** the total number of entries

**Return type** int

**getWords** (*name*)

split the given name into words

**Parameters** **name** (*string*) – the name to split

**Returns** a list of words

**Return type** list

**store** (*sqlDB*)

store my prefix information to the given SQL database

**Parameters** **sqlDB** (*SQLDB*) – the SQL database to use for storing

## 1.7 geograpy.utils module

**geograpy.utils.fuzzy\_match** (*s1, s2, max\_dist=0.8*)

Fuzzy match the given two strings with the given maximum distance :param s1: string: First string :param s2: string: Second string :param max\_dist: float: The distance - default: 0.8

**Returns** jellyfish jaro\_winkler\_similarity based on [https://en.wikipedia.org/wiki/Jaro-Winkler\\_distance](https://en.wikipedia.org/wiki/Jaro-Winkler_distance)

**Return type** float

**geograpy.utils.remove\_non\_ascii** (*s*)

Remove non ascii chars from the given string :param s: string: The string to remove chars from

**Returns** The result string with non-ascii chars removed

**Return type** string

Hat tip: <http://stackoverflow.com/a/1342373/2367526>

## 1.8 Module contents

**geograpy.get\_geoPlace\_context** (*url=None, text=None, debug=False*)

Get a place context for a given text with information about country, region, city and other based on NLTK Named Entities having the Geographic(GPE) label.

**Parameters**

- **url** (*String*) – the url to read text from (if any)
- **text** (*String*) – the text to analyze
- **debug** (*boolean*) – if True show debug information

**Returns** PlaceContext: the place context

**Return type** places

`geograpy.get_place_context` (*url=None, text=None, labels=['GPE', 'GSP', 'PERSON', 'ORGANIZATION'], debug=False*)

Get a place context for a given text with information about country, region, city and other based on NLTK Named Entities in the label set Geographic(GPE), Person(PERSON) and Organization(ORGANIZATION).

**Parameters**

- **url** (*String*) – the url to read text from (if any)
- **text** (*String*) – the text to analyze
- **debug** (*boolean*) – if True show debug information

**Returns** PlaceContext: the place context

**Return type** pc

`geograpy.locate` (*location, correctMisspelling=False, debug=False*)

locate the given location string :param location: the description of the location :type location: string

**Returns** the location

**Return type** *Locator*



## CHAPTER 2

---

setup module

---



## 3.1 Submodules

### 3.2 tests.test\_extractor module

**class** tests.test\_extractor.**TestExtractor** (*methodName='runTest'*)

Bases: unittest.case.TestCase

test Extractor

**check** (*places, expectedList*)

check the places for begin non empty and having at least the expected List of elements

#### Parameters

- **places** (*Places*) – the places to check
- **expectedList** (*list*) – the list of elements to check

**setUp** ()

Hook method for setting up the test fixture before exercising it.

**tearDown** ()

Hook method for deconstructing the test fixture after testing it.

**testExtractorFromText** ()

test different texts for getting geo context information

**testExtractorFromUrl** ()

test the extractor

**testGeograpyIssue32** ()

test <https://github.com/ushahidi/geograpy/issues/32>

**testGetGeoPlace** ()

test geo place handling

```
testIssue10 ()
    test https://github.com/somnathrakshit/geograpy3/issues/10 Add ISO country code

testIssue7 ()
    test https://github.com/somnathrakshit/geograpy3/issues/7 disambiguating countries

testIssue9 ()
    test https://github.com/somnathrakshit/geograpy3/issues/9 [BUG]AttributeError: 'NoneType' object has
    no attribute 'name' on "Pristina, Kosovo"

testStackOverflow54721435 ()
    see https://stackoverflow.com/questions/54721435/unable-to-extract-city-names-from-a-text-using-geograpypython

testStackoverflow43322567 ()
    see https://stackoverflow.com/questions/43322567

testStackoverflow54077973 ()
    see https://stackoverflow.com/questions/54077973/geograpy3-library-for-extracting-the-locations-in-the-text-gives-unicode

testStackoverflow54712198 ()
    see https://stackoverflow.com/questions/54712198/not-only-extracting-places-from-a-text-but-also-other-names-in-geograp

testStackoverflow55548116 ()
    see https://stackoverflow.com/questions/55548116/geograpy3-library-is-not-working-properly-and-give-raise-error

testStackoverflow62152428 ()
    see https://stackoverflow.com/questions/62152428/extracting-country-information-from-description-using-geograpy?noredirect=1#comment112899776\_62152428
```

### 3.3 tests.test\_locator module

Created on 2020-09-19

@author: wf

```
class tests.test_locator.TestLocator (methodName='runTest')
    Bases: unittest.case.TestCase

    test the Locator class from the location module

    setUp ()
        Hook method for setting up the test fixture before exercising it.

    tearDown ()
        Hook method for deconstructing the test fixture after testing it.

    testExamples ()
        test examples

    testGeolite2Cities ()
        test the locs.db cache for cities

    testHasData ()
        check has data and populate functionality

    testIsoRegexp ()
        test regular expression for iso codes

    testWordCount ()
        test the word count
```

### 3.4 tests.test\_places module

```
class tests.test_places.TestPlaces (methodName='runTest')  
    Bases: unittest.case.TestCase  
  
    test Places  
  
    setUp ()  
        Hook method for setting up the test fixture before exercising it.  
  
    tearDown ()  
        Hook method for deconstructing the test fixture after testing it.  
  
    testPlaces ()  
        test places
```

### 3.5 tests.test\_prefixtree module

Created on 2020-09-20

@author: wf

```
class tests.test_prefixtree.TestPrefixTree (methodName='runTest')  
    Bases: unittest.case.TestCase  
  
    test prefix tree algorithm  
  
    setUp ()  
        Hook method for setting up the test fixture before exercising it.  
  
    tearDown ()  
        Hook method for deconstructing the test fixture after testing it.  
  
    testPrefixTree ()  
        test the prefix Tree
```

### 3.6 Module contents



## CHAPTER 4

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



## g

geograpy, 6  
geograpy.extraction, 1  
geograpy.labels, 1  
geograpy.locator, 2  
geograpy.places, 5  
geograpy.prefixtree, 5  
geograpy.utils, 6

## t

tests, 13  
tests.test\_extractor, 11  
tests.test\_locator, 12  
tests.test\_places, 13  
tests.test\_prefixtree, 13



**A**

add() (*geography.prefixtree.PrefixTree method*), 5  
 add2Table() (*geography.prefixtree.PrefixTree method*),  
 5

**C**

check() (*tests.test\_extractor.TestExtractor method*), 11  
 cities\_for\_name() (*geography.locator.Locator  
 method*), 2  
 City (*class in geography.locator*), 2  
 correct\_country\_misspelling() (*ge-  
 ography.locator.Locator method*), 3  
 Country (*class in geography.locator*), 2  
 countStartsWith() (*geography.prefixtree.PrefixTree  
 method*), 6

**D**

db\_has\_data() (*geography.locator.Locator method*), 3  
 default (*geography.labels.Labels attribute*), 2  
 disambiguate() (*geography.locator.Locator method*),  
 3

**E**

Extractor (*class in geography.extraction*), 1

**F**

find\_entities() (*geography.extraction.Extractor  
 method*), 1  
 find\_geoEntities() (*ge-  
 ography.extraction.Extractor method*), 1  
 fromGeoLite2() (*geography.locator.City static  
 method*), 2  
 fromGeoLite2() (*geography.locator.Country static  
 method*), 2  
 fromGeoLite2() (*geography.locator.Region static  
 method*), 5  
 fromPyCountry() (*geography.locator.Country static  
 method*), 2  
 fuzzy\_match() (*in module geography.utils*), 6

**G**

geo (*geography.labels.Labels attribute*), 2  
 geography (*module*), 6  
 geography.extraction (*module*), 1  
 geography.labels (*module*), 1  
 geography.locator (*module*), 2  
 geography.places (*module*), 5  
 geography.prefixtree (*module*), 5  
 geography.utils (*module*), 6  
 get\_geoPlace\_context() (*in module geography*), 6  
 get\_place\_context() (*in module geography*), 7  
 get\_region\_names() (*ge-  
 ography.places.PlaceContext method*), 5  
 getCount() (*geography.prefixtree.PrefixTree method*),  
 6  
 getCountry() (*geography.locator.Locator method*), 3  
 getGeolite2Cities() (*geography.locator.Locator  
 method*), 3  
 getInstance() (*geography.locator.Locator static  
 method*), 3  
 getWords() (*geography.prefixtree.PrefixTree method*),  
 6

**I**

is\_a\_country() (*geography.locator.Locator method*),  
 4  
 isAmbiguousPrefix() (*geography.locator.Locator  
 method*), 3  
 isISO() (*geography.locator.Locator method*), 3  
 isPrefix() (*geography.locator.Locator method*), 4

**L**

Labels (*class in geography.labels*), 2  
 locate() (*geography.locator.Locator method*), 4  
 locate() (*in module geography*), 7  
 Locator (*class in geography.locator*), 2  
 locator (*geography.locator.Locator attribute*), 4

**P**

PlaceContext (*class in geography.places*), 5

places\_by\_name() (*geograpy.locator.Locator method*), 4  
 populate\_Cities() (*geograpy.locator.Locator method*), 4  
 populate\_db() (*geograpy.locator.Locator method*), 4  
 populate\_PrefixAmbiguities() (*geograpy.locator.Locator method*), 4  
 populate\_PrefixTree() (*geograpy.locator.Locator method*), 4  
 PrefixTree (*class in geograpy.prefixtree*), 5

## R

Region (*class in geograpy.locator*), 5  
 regions\_for\_name() (*geograpy.locator.Locator method*), 4  
 remove\_non\_ascii() (*in module geograpy.utils*), 6

## S

set\_cities() (*geograpy.places.PlaceContext method*), 5  
 set\_countries() (*geograpy.places.PlaceContext method*), 5  
 set\_other() (*geograpy.places.PlaceContext method*), 5  
 set\_regions() (*geograpy.places.PlaceContext method*), 5  
 set\_text() (*geograpy.extraction.Extractor method*), 1  
 setAll() (*geograpy.places.PlaceContext method*), 5  
 setUp() (*tests.test\_extractor.TestExtractor method*), 11  
 setUp() (*tests.test\_locator.TestLocator method*), 12  
 setUp() (*tests.test\_places.TestPlaces method*), 13  
 setUp() (*tests.test\_prefixtree.TestPrefixTree method*), 13  
 split() (*geograpy.extraction.Extractor method*), 1  
 store() (*geograpy.prefixtree.PrefixTree method*), 6

## T

tearDown() (*tests.test\_extractor.TestExtractor method*), 11  
 tearDown() (*tests.test\_locator.TestLocator method*), 12  
 tearDown() (*tests.test\_places.TestPlaces method*), 13  
 tearDown() (*tests.test\_prefixtree.TestPrefixTree method*), 13  
 testExamples() (*tests.test\_locator.TestLocator method*), 12  
 TestExtractor (*class in tests.test\_extractor*), 11  
 testExtractorFromText() (*tests.test\_extractor.TestExtractor method*), 11  
 testExtractorFromUrl() (*tests.test\_extractor.TestExtractor method*), 11  
 testGeograpyIssue32() (*tests.test\_extractor.TestExtractor method*), 11  
 testGeolite2Cities() (*tests.test\_locator.TestLocator method*), 12  
 testGetGeoPlace() (*tests.test\_extractor.TestExtractor method*), 11  
 testHasData() (*tests.test\_locator.TestLocator method*), 12  
 testIsoRegexp() (*tests.test\_locator.TestLocator method*), 12  
 testIssue10() (*tests.test\_extractor.TestExtractor method*), 11  
 testIssue7() (*tests.test\_extractor.TestExtractor method*), 12  
 testIssue9() (*tests.test\_extractor.TestExtractor method*), 12  
 TestLocator (*class in tests.test\_locator*), 12  
 TestPlaces (*class in tests.test\_places*), 13  
 testPlaces() (*tests.test\_places.TestPlaces method*), 13  
 TestPrefixTree (*class in tests.test\_prefixtree*), 13  
 testPrefixTree() (*tests.test\_prefixtree.TestPrefixTree method*), 13  
 tests (*module*), 13  
 tests.test\_extractor (*module*), 11  
 tests.test\_locator (*module*), 12  
 tests.test\_places (*module*), 13  
 tests.test\_prefixtree (*module*), 13  
 testStackoverflow43322567() (*tests.test\_extractor.TestExtractor method*), 12  
 testStackoverflow54077973() (*tests.test\_extractor.TestExtractor method*), 12  
 testStackoverflow54712198() (*tests.test\_extractor.TestExtractor method*), 12  
 testStackoverflow54721435() (*tests.test\_extractor.TestExtractor method*), 12  
 testStackoverflow55548116() (*tests.test\_extractor.TestExtractor method*), 12  
 testStackoverflow62152428() (*tests.test\_extractor.TestExtractor method*), 12  
 testWordCount() (*tests.test\_locator.TestLocator method*), 12