

---

# geograpy3

Jun 21, 2021



---

## 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	8
1.6	geograpy.prefixtree module	8
1.7	geograpy.utils module	8
1.8	geograpy.wikidata module	9
1.9	Module contents	10
<b>2</b>	<b>setup module</b>	<b>11</b>
<b>3</b>	<b>tests package</b>	<b>13</b>
3.1	Submodules	13
3.2	tests.test_extractor module	13
3.3	tests.test_locator module	14
3.4	tests.test_places module	15
3.5	tests.test_prefixtree module	16
3.6	tests.test_wikidata module	16
3.7	Module contents	16
<b>4</b>	<b>Indices and tables</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>
	<b>Index</b>	<b>21</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** (*delimiter=', '*)

simpler regular expression splitter with not entity check

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

## 1.3 geograpy.labels module

@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(**kwargs)
    Bases: geograpy.locator.Location

    a single city as an object

    static fromGeoLite2(record)

    classmethod getSamples()

    setValue(name, record)
        set a field value with the given name to the given record dicts corresponding entry or none
```

### Parameters

- **name** (*string*) – the name of the field
- **record** (*dict*) – the dict to get the value from

```
class geograpy.locator.CityList
    Bases: geograpy.locator.LocationList

    a list of cities

    classmethod fromJSONBackup()
        get city list from json backup (json backup is based on wikidata query results)

        Returns CityList based on the json backup

    classmethod fromWikidata(fromBackup: bool = True, countryIDs: list = None, regionIDs: list = None)
        get city list form wikidata
```

**Parameters**

- **fromBackup** (*bool*) – If True instead of querying wikidata a backup of the wikidata results is used to create the city list. Otherwise wikidata is queried for the city data. Default is True
- **countryIDs** (*list*) – List of countryWikiDataIDs. Limits the returned cities to the given countries
- **regionIDs** (*list*) – List of regionWikiDataIDs. Limits the returned cities to the given regions

**Returns** CityList based wikidata query results

```
class geograsy.locator.Country (lookupSource='sqlDB', **kwargs)
```

Bases: *geograsy.locator.Location*

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*

```
classmethod getSamples ()
```

```
class geograsy.locator.CountryList
```

Bases: *geograsy.locator.LocationList*

a list of countries

```
classmethod fromErdem ()
```

get country list provided by Erdem Ozkol <https://github.com/erdem>

```
classmethod fromJSONBackup ()
```

get country list from json backup (json backup is based on wikidata query results)

**Returns** CountryList based on the json backup

```
classmethod fromWikidata ()
```

get country list form wikidata

```
classmethod from_sqlDb (sqlDB)
```

```
class geograsy.locator.Earth
```

Bases: object

```
radius = 6371.0
```

```
class geograsy.locator.Location (**kwargs)
```

Bases: *lodstorage.jsonable.JSONable*

Represents a Location

```
balltreeQueryResultToLocationList (distances, indices, lookupListOfLocations)
```

convert the given ballTree Query Result to a LocationList

**Parameters**

- **distances** (*list*) – array of distances

- **indices** (*list*) – array of indices
- **lookupListOfLocations** (*list*) – a list of valid locations to use for lookup

**Returns** a list of result Location/distance tuples

**Return type** list

**distance** (*other*) → float

calculate the distance to another Location

**Parameters** **other** (*Location*) – the other location

**Returns** the haversine distance in km

**getLocationsWithinRadius** (*lookupLocationList, radiusKm: float*)

Gives the n closest locations to me from the given lookupListOfLocations

**Parameters**

- **lookupLocationList** (*LocationList*) – a LocationList object to use for lookup
- **radiusKm** (*float*) – the radius in which to check (in km)

**Returns** a list of result Location/distance tuples

**Return type** list

**getNClosestLocations** (*lookupLocationList, n: int*)

Gives a list of up to n locations which have the shortest distance to me as calculated from the given listOfLocations

**Parameters**

- **lookupLocationList** (*LocationList*) – a LocationList object to use for lookup
- **n** (*int*) – the maximum number of closest locations to return

**Returns** a list of result Location/distance tuples

**Return type** list

**classmethod** **getSamples** ()

**static** **haversine** (*lon1, lat1, lon2, lat2*)

Calculate the great circle distance between two points on the earth (specified in decimal degrees)

**class** geograpy.locator.**LocationList** (*listName: str = None, clazz=None, tableName: str = None*)

Bases: lodstorage.jsonable.JSONAbleList

a list of locations

**getBallTuple** (*cache: bool = True*)

get the BallTuple=BallTree,validList of this location list

**Parameters**

- **cache** (*bool*) – if True calculate and use a cached version otherwise recalculate on
- **call of this function** (*every*) –

**Returns** a sklearn.neighbors.BallTree for the given list of locations, list: the valid list of locations list: valid list of locations

**Return type** BallTree,list

**getLocationByID** (*wikidataID: str*)

Returns the location object that corresponds to the given location



**Parameters** `wikidataID` – wikidataid of the location that should be returned

**Returns** Location object

**getLocationList** ()

get my location list

**static getURLContent** (*url: str*)

**class** `geograpy.locator.Locator` (*db\_file=None, correctMisspelling=False, debug=False*)

Bases: object

location handling

**cities\_for\_name** (*cityName*)

find cities with the given cityName

**Parameters** `cityName` (*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

**createViews** (*sqlDB*)

**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

**db\_recordCount** (*tableList, tableName*)

count the number of records for the given tableName

**Parameters**

- **tableList** (*list*) – the list of table to check
- **tableName** (*str*) – the name of the table to check

**Returns** int: the number of records found for the table

**disambiguate** (*country, regions, cities, byPopulation=True*)

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*

**getAliases** ()

get the aliases hashTable

**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

**getView** ()

get the view to be used

**Returns** the SQL view to be used for CityLookups e.g. GeoLite2CityLookup

**Return type** str

**getWikidataCityPopulation** (*sqlDB, endpoint=None*)**Parameters**

- **sqlDB** (*SQLDB*) – target SQL database
- **endpoint** (*str*) – url of the wikidata endpoint or None if default should be used

**isISO** (*s*)

check if the given string is an ISO code

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

**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

**locateCity** (*places*)

locate a city, region country combination based on the given wordtoken information

**Parameters**

- **places** (*list*) – a list of places derived by splitting a locality e.g. “San Francisco, CA”
- **to** **"San Francisco", "CA"** (*leads*) –

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

**Return type** *City*

**locator** = None

**places\_by\_name** (*placeName, columnName*)  
 get places by name and column :param placeName: the name of the place :type placeName: string :param columnName: the column to look at :type columnName: string

**populateFromWikidata** (*sqlDB*)  
 populate countries and regions from Wikidata  
**Parameters** **sqlDB** (*SQLDB*) – target SQL database

**populate\_Cities** (*sqlDB*)  
 populate the given sqlDB with the Geolite2 Cities  
**Parameters** **sqlDB** (*SQLDB*) – the SQL database to use

**populate\_Cities\_FromWikidata** (*sqlDB*)  
 populate the given sqlDB with the Wikidata Cities  
**Parameters** **sqlDB** (*SQLDB*) – target SQL database

**populate\_Countries** (*sqlDB*)  
 populate database with countries from wikiData  
**Parameters** **sqlDB** (*SQLDB*) – target SQL database

**populate\_Regions** (*sqlDB*)  
 populate database with regions from wikiData  
**Parameters** **sqlDB** (*SQLDB*) – target SQL database

**populate\_Version** (*sqlDB*)  
 populate the version table  
**Parameters** **sqlDB** (*SQLDB*) – target SQL database

**populate\_db** (*force=False*)  
 populate the cities SQL database which caches the information from the GeoLite2-City-Locations.csv file  
**Parameters** **force** (*bool*) – if True force a recreation of the database

**readCSV** (*fileName*)

**recreateDatabase** ()  
 recreate my lookup database

**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

**static resetInstance** ()

**class** geograpy.locator.**Region** (*\*\*kwargs*)  
 Bases: *geograpy.locator.Location*  
 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*

**static fromWikidata** (*record*)  
create a region from a Wikidata record

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

**Returns** the corresponding region information

**Return type** *Region*

**classmethod** **getSamples** ()

**class** `geograpy.locator.RegionList`  
Bases: *geograpy.locator.LocationList*  
a list of regions

**classmethod** **fromJSONBackup** ()  
get region list from json backup (json backup is based on wikidata query results)

**Returns** RegionList based on the json backup

**classmethod** **fromWikidata** ()  
get region list form wikidata

**classmethod** **from\_sqlDb** (*sqlDB*)

`geograpy.locator.main` (*argv=None*)  
main program.

## 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

## 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 geograpy.wikidata module

Created on 2020-09-23

@author: wf

**class** `geograpy.wikidata.Wikidata` (*endpoint='https://query.wikidata.org/sparql'*)

Bases: `object`

Wikidata access

**getCities** (*region=None, country=None*)

get the cities from Wikidata

**Parameters**

- **region** – List of country Wikidata IDs. Limits the returned cities to the given countries
- **country** – List of region Wikidata IDs. Limits the returned cities to the given regions

**getCitiesOfRegion** (*regionWikidataId: str, limit: int*)

Queries the cities of the given region. If the region is a city state the region is returned as city. The cities are ordered by population and can be limited by the given limit attribute.

**Parameters**

- **regionWikidataId** – wikidata id of the region the cities should be queried for
- **limit** – Limits the amount of returned cities

**Returns** Returns list of cities of the given region ordered by population

**getCityPopulations** (*profile=True*)

get the city populations from Wikidata

**Parameters** **profile** (*bool*) – if True show profiling information

**static** **getCoordinateComponents** (*coordinate: str*) -> (*<class 'float'>*, *<class 'float'>*)

Converts the wikidata coordinate representation into its subcomponents longitude and latitude Example: 'Point(-118.25 35.05694444)' results in ('-118.25' '35.05694444')

**Parameters** **coordinate** – coordinate value in the format as returned by wikidata queries

**Returns** Returns the longitude and latitude of the given coordinate as separate values

**getCountries** ()

get a list of countries

*try query*

**getRegions()**  
get Regions from Wikidata

try query

**static getValuesClause** (*varName: str, values, wikidataEntities: bool = True*)  
generates the SPARQL value clause for the given variable name containing the given values :param varName: variable name for the ValuesClause :param values: values for the clause :param wikidataEntities: if true the wikidata prefix is added to the values otherwise it is expected taht the given values are proper IRIs :type wikidataEntities: bool

**Returns** str

**static getWikidataId** (*wikidataURL: str*)  
Extracts the wikidata id from the given wikidata URL

**Parameters** **wikidataURL** – wikidata URL the id should be extracted from

**Returns** The wikidata id if present in the given wikidata URL otherwise None

## 1.9 Module contents

main geograpy 3 module

**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.locateCity** (*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-traceback-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

    checkExamples (examples, countries, debug=False, check=True)
        check that the given example give results in the given countries :param examples: a list of example location
        strings :type examples: list :param countries: a list of expected country iso codes :type countries: list

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

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

    testDelimiters ()
        test the delimiter statistics for names

    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

testIssue15 ()
    https://github.com/somnathrakshit/geograpy3/issues/15 test Issue 15 Disambiguate via population, gdp
    data

testIssue17 ()
    test issue 17:

    https://github.com/somnathrakshit/geograpy3/issues/17

    [BUG] San Francisco, USA and Auckland, New Zealand should be locatable #17

testIssue19 ()
    test issue 19

testIssue22 ()
    https://github.com/somnathrakshit/geograpy3/issues/22

testIssue41_CountriesFromErdem ()
    test getting Country list from Erdem

testIssue_42_distance ()
    test haversine and location

testPopulation ()
    test adding population data from wikidata to GeoLite2 information

testProceedingsExample ()
    test a proceedings title Example

testStackOverflow64379688 ()
    compare old and new geograpy interface

testStackOverflow64418919 ()
    https://stackoverflow.com/questions/64418919/problem-retrieving-region-in-us-with-geograpy3

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

## 3.6 tests.test\_wikidata module

Created on 2020-09-23

@author: wf

```
class tests.test_wikidata.TestWikidata (methodName='runTest')
    Bases: unittest.case.TestCase
        test the wikidata access for cities

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

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

    testGetCitiesOfRegion ()
        Test getting cities based on region wikidata id

    testGetCoordinateComponents ()
        test the splitting of coordinate components in WikiData query results

    testGetWikidataId ()

    testLocatorWithWikiData ()
        test Locator

    testWikidataCities ()
        test getting city information from wikidata
        1372 Singapore 749 Beijing, China 704 Paris, France 649 Barcelona, Spain 625 Rome, Italy 616 Hong
        Kong 575 Bangkok, Thailand 502 Vienna, Austria 497 Athens, Greece 483 Shanghai, China

    testWikidataCountries ()
        test getting country information from wikidata
```

## 3.7 Module contents

## CHAPTER 4

---

### Indices and tables

---

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



### g

- `geograpy`, [10](#)
- `geograpy.extraction`, [1](#)
- `geograpy.labels`, [1](#)
- `geograpy.locator`, [2](#)
- `geograpy.places`, [8](#)
- `geograpy.utils`, [8](#)
- `geograpy.wikidata`, [9](#)

### t

- `tests`, [16](#)
- `tests.test_extractor`, [13](#)
- `tests.test_locator`, [14](#)
- `tests.test_places`, [15](#)
- `tests.test_wikidata`, [16](#)





## B

balltreeQueryResultToLocationList() (*geography.locator.Location* method), 3

## C

check() (*tests.test\_extractor.TestExtractor* method), 13

checkExamples() (*tests.test\_locator.TestLocator* method), 14

cities\_for\_name() (*geography.locator.Locator* method), 5

City (class in *geography.locator*), 2

CityList (class in *geography.locator*), 2

correct\_country\_misspelling() (*geography.locator.Locator* method), 5

Country (class in *geography.locator*), 3

CountryList (class in *geography.locator*), 3

createViews() (*geography.locator.Locator* method), 5

## D

db\_has\_data() (*geography.locator.Locator* method), 5

db\_recordCount() (*geography.locator.Locator* method), 5

default (*geography.labels.Labels* attribute), 2

disambiguate() (*geography.locator.Locator* method), 5

distance() (*geography.locator.Location* method), 4

## E

Earth (class in *geography.locator*), 3

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

## F

find\_entities() (*geography.extraction.Extractor* method), 1

find\_geoEntities() (*geography.extraction.Extractor* method), 1

from\_sqlDb() (*geography.locator.CountryList* class method), 3

from\_sqlDb() (*geography.locator.RegionList* class method), 8

fromErdem() (*geography.locator.CountryList* class method), 3

fromGeoLite2() (*geography.locator.City* static method), 2

fromGeoLite2() (*geography.locator.Country* static method), 3

fromGeoLite2() (*geography.locator.Region* static method), 7

fromJSONBackup() (*geography.locator.CityList* class method), 2

fromJSONBackup() (*geography.locator.CountryList* class method), 3

fromJSONBackup() (*geography.locator.RegionList* class method), 8

fromPyCountry() (*geography.locator.Country* static method), 3

fromWikidata() (*geography.locator.CityList* class method), 2

fromWikidata() (*geography.locator.CountryList* class method), 3

fromWikidata() (*geography.locator.Region* static method), 8

fromWikidata() (*geography.locator.RegionList* class method), 8

fuzzy\_match() (in module *geography.utils*), 8

## G

geo (*geography.labels.Labels* attribute), 2

geography (module), 10

geography.extraction (module), 1

geography.labels (module), 1

geography.locator (module), 2

geography.places (module), 8

geography.utils (module), 8

geography.wikidata (module), 9

get\_geoPlace\_context() (in module *geography*), 10

get\_place\_context() (in module *geography*), 10

- get\_region\_names() (geography.places.PlaceContext method), 8  
 getAliases() (geography.locator.Locator method), 5  
 getBallTuple() (geography.locator.LocationList method), 4  
 getCities() (geography.wikidata.Wikidata method), 9  
 getCitiesOfRegion() (geography.wikidata.Wikidata method), 9  
 getCityPopulations() (geography.wikidata.Wikidata method), 9  
 getCoordinateComponents() (geography.wikidata.Wikidata static method), 9  
 getCountries() (geography.wikidata.Wikidata method), 9  
 getCountry() (geography.locator.Locator method), 5  
 getGeolite2Cities() (geography.locator.Locator method), 6  
 getInstance() (geography.locator.Locator static method), 6  
 getLocationByID() (geography.locator.LocationList method), 4  
 getLocationList() (geography.locator.LocationList method), 5  
 getLocationsWithinRadius() (geography.locator.Location method), 4  
 getNClosestLocations() (geography.locator.Location method), 4  
 getRegions() (geography.wikidata.Wikidata method), 9  
 getSamples() (geography.locator.City class method), 2  
 getSamples() (geography.locator.Country class method), 3  
 getSamples() (geography.locator.Location class method), 4  
 getSamples() (geography.locator.Region class method), 8  
 getURLContent() (geography.locator.LocationList static method), 5  
 getValuesClause() (geography.wikidata.Wikidata static method), 10  
 getView() (geography.locator.Locator method), 6  
 getWikidataCityPopulation() (geography.locator.Locator method), 6  
 getWikidataId() (geography.wikidata.Wikidata static method), 10
- ## H
- haversine() (geography.locator.Location static method), 4
- ## I
- is\_a\_country() (geography.locator.Locator method), 6
- ## L
- Labels (class in geography.labels), 2  
 locateCity() (geography.locator.Locator method), 6  
 locateCity() (in module geography), 10  
 Location (class in geography.locator), 3  
 LocationList (class in geography.locator), 4  
 Locator (class in geography.locator), 5  
 locator (geography.locator.Locator attribute), 6
- ## M
- main() (in module geography.locator), 8
- ## P
- PlaceContext (class in geography.places), 8  
 places\_by\_name() (geography.locator.Locator method), 6  
 populate\_Cities() (geography.locator.Locator method), 7  
 populate\_Cities\_FromWikidata() (geography.locator.Locator method), 7  
 populate\_Countries() (geography.locator.Locator method), 7  
 populate\_db() (geography.locator.Locator method), 7  
 populate\_Regions() (geography.locator.Locator method), 7  
 populate\_Version() (geography.locator.Locator method), 7  
 populateFromWikidata() (geography.locator.Locator method), 7
- ## R
- radius (geography.locator.Earth attribute), 3  
 readCSV() (geography.locator.Locator method), 7  
 recreateDatabase() (geography.locator.Locator method), 7  
 Region (class in geography.locator), 7  
 RegionList (class in geography.locator), 8  
 regions\_for\_name() (geography.locator.Locator method), 7  
 remove\_non\_ascii() (in module geography.utils), 9  
 resetInstance() (geography.locator.Locator static method), 7
- ## S
- set\_cities() (geography.places.PlaceContext method), 8  
 set\_countries() (geography.places.PlaceContext method), 8  
 set\_other() (geography.places.PlaceContext method), 8

set\_regions() (geograpy.places.PlaceContext method), 8  
 set\_text() (geograpy.extraction.Extractor method), 1  
 setAll() (geograpy.places.PlaceContext method), 8  
 setUp() (tests.test\_extractor.TestExtractor method), 13  
 setUp() (tests.test\_locator.TestLocator method), 14  
 setUp() (tests.test\_places.TestPlaces method), 15  
 setUp() (tests.test\_wikidata.TestWikidata method), 16  
 setValue() (geograpy.locator.City method), 2  
 split() (geograpy.extraction.Extractor method), 1

## T

tearDown() (tests.test\_extractor.TestExtractor method), 13  
 tearDown() (tests.test\_locator.TestLocator method), 14  
 tearDown() (tests.test\_places.TestPlaces method), 15  
 tearDown() (tests.test\_wikidata.TestWikidata method), 16  
 testDelimiters() (tests.test\_locator.TestLocator method), 14  
 testExamples() (tests.test\_locator.TestLocator method), 14  
 TestExtractor (class in tests.test\_extractor), 13  
 testExtractorFromText() (tests.test\_extractor.TestExtractor method), 13  
 testExtractorFromUrl() (tests.test\_extractor.TestExtractor method), 13  
 testGeograpyIssue32() (tests.test\_extractor.TestExtractor method), 13  
 testGeolite2Cities() (tests.test\_locator.TestLocator method), 14  
 testGetCitiesOfRegion() (tests.test\_wikidata.TestWikidata method), 16  
 testGetCoordinateComponents() (tests.test\_wikidata.TestWikidata method), 16  
 testGetGeoPlace() (tests.test\_extractor.TestExtractor method), 13  
 testGetWikidataId() (tests.test\_wikidata.TestWikidata method), 16  
 testHasData() (tests.test\_locator.TestLocator method), 14  
 testIsoRegexp() (tests.test\_locator.TestLocator method), 14  
 testIssue10() (tests.test\_extractor.TestExtractor method), 13  
 testIssue15() (tests.test\_locator.TestLocator method), 15  
 testIssue17() (tests.test\_locator.TestLocator method), 15  
 testIssue19() (tests.test\_locator.TestLocator method), 15  
 testIssue22() (tests.test\_locator.TestLocator method), 15  
 testIssue41\_CountriesFromErdem() (tests.test\_locator.TestLocator method), 15  
 testIssue7() (tests.test\_extractor.TestExtractor method), 14  
 testIssue9() (tests.test\_extractor.TestExtractor method), 14  
 testIssue\_42\_distance() (tests.test\_locator.TestLocator method), 15  
 TestLocator (class in tests.test\_locator), 14  
 testLocatorWithWikiData() (tests.test\_wikidata.TestWikidata method), 16  
 TestPlaces (class in tests.test\_places), 15  
 testPlaces() (tests.test\_places.TestPlaces method), 15  
 testPopulation() (tests.test\_locator.TestLocator method), 15  
 testProceedingsExample() (tests.test\_locator.TestLocator method), 15  
 tests (module), 16  
 tests.test\_extractor (module), 13  
 tests.test\_locator (module), 14  
 tests.test\_places (module), 15  
 tests.test\_wikidata (module), 16  
 testStackoverflow43322567() (tests.test\_extractor.TestExtractor method), 14  
 testStackoverflow54077973() (tests.test\_extractor.TestExtractor method), 14  
 testStackoverflow54712198() (tests.test\_extractor.TestExtractor method), 14  
 testStackoverflow54721435() (tests.test\_extractor.TestExtractor method), 14  
 testStackoverflow55548116() (tests.test\_extractor.TestExtractor method), 14  
 testStackoverflow62152428() (tests.test\_extractor.TestExtractor method), 14  
 testStackoverflow64379688() (tests.test\_locator.TestLocator method), 15  
 testStackoverflow64418919() (tests.test\_locator.TestLocator method), 15

TestWikidata (*class in tests.test\_wikidata*), 16  
testWikidataCities ()  
    (*tests.test\_wikidata.TestWikidata*      *method*),  
    16  
testWikidataCountries ()  
    (*tests.test\_wikidata.TestWikidata*      *method*),  
    16  
testWordCount ()      (*tests.test\_locator.TestLocator*  
    *method*), 15

## W

Wikidata (*class in geograpy.wikidata*), 9