
geograpy3

Oct 10, 2020

Contents:

1	geograpy package	1
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	6
1.6	geograpy.prefixtree module	6
1.7	geograpy.utils module	6
1.8	geograpy.wikidata module	6
1.9	Module contents	7
2	setup module	9
3	tests package	11
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	tests.test_wikidata module	13
3.7	Module contents	14
4	Indices and tables	15
	Python Module Index	17
	Index	19

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
    Bases: object

    a single city as an object

    static fromGeoLite2(record)

    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.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 (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**
 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*

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*

`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
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

    getCityPopulations (profile=True)
        get the city populations from Wikidata

        Parameters profile (bool) – if True show profiling information

    getCountries ()
        get a list of countries

        try query

    getRegions ()
        get Regions from Wikidata

        try query

```

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

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

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.

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`, [7](#)
- `geograpy.extraction`, [1](#)
- `geograpy.labels`, [1](#)
- `geograpy.locator`, [2](#)
- `geograpy.places`, [6](#)
- `geograpy.utils`, [6](#)
- `geograpy.wikidata`, [6](#)

t

- `tests`, [14](#)
- `tests.test_extractor`, [11](#)
- `tests.test_locator`, [12](#)
- `tests.test_places`, [13](#)
- `tests.test_wikidata`, [13](#)

C

check() (*tests.test_extractor.TestExtractor method*), 11
 checkExamples() (*tests.test_locator.TestLocator method*), 12
 cities_for_name() (*geography.locator.Locator method*), 3
 City (*class in geography.locator*), 2
 correct_country_misspelling() (*geography.locator.Locator method*), 3
 Country (*class in geography.locator*), 2
 createViews() (*geography.locator.Locator method*), 3

D

db_has_data() (*geography.locator.Locator method*), 3
 db_recordCount() (*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() (*geography.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
 fromWikidata() (*geography.locator.Region static method*), 5
 fuzzy_match() (*in module geography.utils*), 6

G

geo (*geography.labels.Labels attribute*), 2
 geography (*module*), 7
 geography.extraction (*module*), 1
 geography.labels (*module*), 1
 geography.locator (*module*), 2
 geography.places (*module*), 6
 geography.utils (*module*), 6
 geography.wikidata (*module*), 6
 get_geoPlace_context() (*in module geography*), 7
 get_place_context() (*in module geography*), 7
 get_region_names() (*geography.places.PlaceContext method*), 6
 getAliases() (*geography.locator.Locator method*), 3
 getCities() (*geography.wikidata.Wikidata method*), 7
 getCityPopulations() (*geography.wikidata.Wikidata method*), 7
 getCountries() (*geography.wikidata.Wikidata method*), 7
 getCountry() (*geography.locator.Locator method*), 3
 getGeolite2Cities() (*geography.locator.Locator method*), 3
 getInstance() (*geography.locator.Locator static method*), 4
 getRegions() (*geography.wikidata.Wikidata method*), 7
 getView() (*geography.locator.Locator method*), 4
 getWikidataCityPopulation() (*geography.locator.Locator method*), 4

I

is_a_country() (*geography.locator.Locator method*), 4
 isISO() (*geography.locator.Locator method*), 4

L

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

Locator (*class in geograpy.locator*), 2
locator (*geograpy.locator.Locator attribute*), 4

M

main() (*in module geograpy.locator*), 5

P

PlaceContext (*class in geograpy.places*), 6
places_by_name() (*geograpy.locator.Locator method*), 4
populate_Cities() (*geograpy.locator.Locator method*), 4
populate_Cities_FromWikidata() (*geograpy.locator.Locator method*), 5
populate_Countries() (*geograpy.locator.Locator method*), 5
populate_db() (*geograpy.locator.Locator method*), 5
populate_Regions() (*geograpy.locator.Locator method*), 5
populate_Version() (*geograpy.locator.Locator method*), 5
populateFromWikidata() (*geograpy.locator.Locator method*), 4

R

readCSV() (*geograpy.locator.Locator method*), 5
recreateDatabase() (*geograpy.locator.Locator method*), 5
Region (*class in geograpy.locator*), 5
regions_for_name() (*geograpy.locator.Locator method*), 5
remove_non_ascii() (*in module geograpy.utils*), 6
resetInstance() (*geograpy.locator.Locator static method*), 5

S

set_cities() (*geograpy.places.PlaceContext method*), 6
set_countries() (*geograpy.places.PlaceContext method*), 6
set_other() (*geograpy.places.PlaceContext method*), 6
set_regions() (*geograpy.places.PlaceContext method*), 6
set_text() (*geograpy.extraction.Extractor method*), 1
setAll() (*geograpy.places.PlaceContext method*), 6
setUp() (*tests.test_extractor.TestExtractor method*), 11
setUp() (*tests.test_locator.TestLocator method*), 12
setUp() (*tests.test_places.TestPlaces method*), 13
setUp() (*tests.test_wikidata.TestWikidata method*), 13
setValue() (*geograpy.locator.City method*), 2
split() (*geograpy.extraction.Extractor method*), 1

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_wikidata.TestWikidata method*), 13
testDelimiters() (*tests.test_locator.TestLocator method*), 12
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
testIssue15() (*tests.test_locator.TestLocator method*), 13
testIssue17() (*tests.test_locator.TestLocator method*), 13
testIssue19() (*tests.test_locator.TestLocator method*), 13
testIssue22() (*tests.test_locator.TestLocator method*), 13
testIssue7() (*tests.test_extractor.TestExtractor method*), 12
testIssue9() (*tests.test_extractor.TestExtractor method*), 12
TestLocator (*class in tests.test_locator*), 12
testLocatorWithWikiData() (*tests.test_wikidata.TestWikidata method*), 14
TestPlaces (*class in tests.test_places*), 13
testPlaces() (*tests.test_places.TestPlaces method*), 13
testPopulation() (*tests.test_locator.TestLocator method*), 13

[tests \(module\)](#), 14
[tests.test_extractor \(module\)](#), 11
[tests.test_locator \(module\)](#), 12
[tests.test_places \(module\)](#), 13
[tests.test_wikidata \(module\)](#), 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
[TestWikidata \(class in tests.test_wikidata\)](#), 13
[testWikidataCities \(\)](#)
 ([tests.test_wikidata.TestWikidata](#) *method*),
 14
[testWikidataCountries \(\)](#)
 ([tests.test_wikidata.TestWikidata](#) *method*),
 14
[testWordCount \(\)](#) ([tests.test_locator.TestLocator](#)
 method), 13

W

[Wikidata \(class in geograpy.wikidata\)](#), 6