
geography3

Apr 08, 2023

Contents:

1	geography package	1
1.1	Submodules	1
1.2	geography.extraction module	1
1.3	geography.labels module	1
1.4	geography.locator module	2
1.5	geography.places module	9
1.6	geography.prefixtree module	10
1.7	geography.utils module	10
1.8	geography.wikidata module	11
1.9	Module contents	12
2	setup module	13
3	tests package	15
3.1	Submodules	15
3.2	tests.test_extractor module	15
3.3	tests.test_locator module	16
3.4	tests.test_places module	17
3.5	tests.test_prefixtree module	18
3.6	tests.test_wikidata module	18
3.7	Module contents	18
4	Indices and tables	19
	Python Module Index	21
	Index	23

CHAPTER 1

geography package

1.1 Submodules

1.2 geography.extraction module

```
class geography.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 no entity check
    hat tip: https://stackoverflow.com/a/1059601/1497139
```

1.3 geography.labels module

Created on 2020-09-10

```
@author: wf

class geography.labels.Labels
    Bases: object

    NLTK labels

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

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

1.4 geography.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 geography.locator.City(**kwargs)
    Bases: geography.locator.Location

    a single city as an object

    country

    static fromCityLookup(cityLookupRecord: dict)
        create a city from a cityLookupRecord and setting City, Region and Country while at it :param cityRecord:
        a map derived from the CityLookup view :type cityRecord: dict

    classmethod getSamples()

    region

    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 geography.locator.CityManager(name: str = 'CityManager', config: lodstorage.storageconfig.StorageConfig = None, debug=False)
    Bases: geography.locator.LocationManager
```

a list of cities

```

classmethod getJsonFiles (config: lodstorage.storageconfig.StorageConfig) → list
    get the list of the json files that have my data

    Returns a list of json file names

    Return type list

class geography.locator.Country (lookupSource='sqlDB', **kwargs)
    Bases: geography.locator.Location

    a country

    static fromCountryLookup (countryLookupRecord: dict)
        create a region from a regionLookupRecord and setting Region and Country while at it :param regionRecord: a map derived from the CityLookup view :type regionRecord: dict

    classmethod getSamples ()

class geography.locator.CountryManager (name: str = 'CountryManager', config: lodstorage.storageconfig.StorageConfig = None, debug=False)
    Bases: geography.locator.LocationManager

    a list of countries

    classmethod fromErdem ()
        get country list provided by Erdem Ozkol https://github.com/erdem

class geography.locator.Earth
    Bases: object

    radius = 6371.0

class geography.locator.Location (**kwargs)
    Bases: lodstorage.jsonable.JSONAble

    Represents a Location

    balltreeQueryResultToLocationManager (distances, indices, lookupListOfLocations)
        convert the given ballTree Query Result to a LocationManager

        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

    classmethod fromRecord (regionRecord: dict)
        create a location from a dict record

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

        Returns the corresponding region information

        Return type Region

```

getLocationsWithinRadius (*lookupLocationManager, radiusKm: float*)

Gives the n closest locations to me from the given lookupListOfLocations

Parameters

- **lookupLocationManager** ([LocationManager](#)) – a LocationManager 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 (*lookupLocationManager, n: int*)

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

Parameters

- **lookupLocationManager** ([LocationManager](#)) – a LocationManager 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)

isKnownAs (*name*) → bool

Checks if this location is known under the given name

Parameters **name** (*str*) – name the location should be checked against

Returns True if the given name is either the name of the location or present in the labels of the location

static mappedDict (*record, keyMapList: list*)

static partialDict (*record, clazz, keys=None*)

class [geography.locator.LocationContext](#) (*countryManager: geography.locator.CountryManager, regionManager: geography.locator.RegionManager, cityManager: geography.locator.CityManager, config: lodstorage.storageconfig.StorageConfig*)

Bases: object

Holds LocationManagers of all hierarchy levels and provides methods to traverse through the levels

cities

countries

db_filename = 'locations.db'

classmethod fromCache (*config: lodstorage.storageconfig.StorageConfig = None, forceUpdate: bool = False*)

Init a LocationContext from Cache if existent otherwise init cache

Parameters

- **config** (*StorageConfig*) – configuration of the cache if None the default config is used
- **forceUpdate** (*bool*) – If True an existent cache will be over written

static getDefaultConfig() → *lodstorage.storageconfig.StorageConfig*
Returns default StorageConfig

interlinkLocations (*warnOnDuplicates: bool = True, profile=True*)
Interlinks locations by adding the hierarchy references to the locations

Parameters **warnOnDuplicates** (*bool*) – if there are duplicates warn

load (*forceUpdate: bool = False, warnOnDuplicates: bool = False*)
load my data

locateLocation (**locations*, *verbose: bool = False*)

Get possible locations for the given location names. Current prioritization of the results is city(ordered by population)→region→country ToDo: Extend the ranking of the results e.g. matching of multiple location parts increase ranking :param *locations: :param verbose: If True combinations of locations names are used to improve the search results. (Increases lookup time) :type verbose: bool

Returns:

regions

class *geography.locator.LocationManager* (*name: str, entityName: str, entityPluralName: str, listName: str = None, tableName: str = None, clazz=None, primaryKey: str = None, config: lodstorage.storageconfig.StorageConfig = None, handleInvalidListTypes=True, filterInvalidListTypes=False, debug=False*)

Bases: *lodstorage.entity.EntityManager*

a list of locations

add (*location*)

add the given location to me

Parameters **location** (*object*) – the location to be added and put in my hash map

classmethod downloadBackupFileFromGitHub (*fileName: str, targetDirectory: str = None, force: bool = False*)

download the given fileName from the github data directory

Parameters

- **fileName** (*str*) – the filename to download
- **targetDirectory** (*str*) – download the file this directory
- **force** (*bool*) – force the overwriting of the existent file

Returns the local file

Return type *str*

fromCache (*force=False, getListOfDicts=None, sampleRecordCount=-1*)
get me from the cache

static getBackupDirectory()

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

getByName (**names*)

Get locations matching given names :param name: Name of the location

Returns Returns locations that match the given name

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

getLocationByIsoCode (*isoCode*: str)

Get possible locations matching the given isoCode :param isoCode: isoCode of possible Locations

Returns List of wikidata ids of locations matching the given isoCode

getLocationsByWikidataId(**wikidataId*)

Returns Location objects for the given wikidataids :param *wikidataId: wikidataIds of the locations that should be returned :type *wikidataId: str

Returns Location objects matching the given wikidataids

class geography.locator.**Locator** (*db_file=None*, *correctMisspelling=False*, *storageConfig: lod-storage.storageconfig.StorageConfig = None*, *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*

downloadDB (*forceUpdate: bool = False*)
download my database

Parameters **forceUpdate** (*bool*) – force the overwriting of the existent file

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*

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. CityLookup

Return type *str*

static isISO (*s*)

check if the given string is an ISO code (ISO 3166-2 code) see <https://www.wikidata.org/wiki/Property:P300>

Returns True if the string might be an ISO Code as per a regexp check

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*

loadDB()
loads the database from cache and sets it as sqlDB property

locateCity(places: list)
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

normalizePlaces(places: list)
normalize places

Parameters **places** (*list*) –

Returns stripped and aliased list of places

Return type list

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

populate_Cities(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: str)
read the given CSV file

Parameters **fileName** (*str*) – the filename to read

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 geography.locator.Region(**kwargs)
    Bases: geography.locator.Location
    a Region (Subdivision)
```

country

```
static fromRegionLookup(regionLookupRecord: dict)
    create a region from a regionLookupRecord and setting Region and Country while at it :param regionRecord: a map derived from the CityLookup view :type regionRecord: dict
```

```
classmethod getSamples()
```

```
class geography.locator.RegionManager(name: str = 'RegionManager', config: lodstorage.storageconfig.StorageConfig = None, debug=False)
    Bases: geography.locator.LocationManager
    a list of regions
```

```
geography.locator.main(argv=None)
    main program.
```

1.5 geography.places module

```
class geography.places.PlaceContext(place_names: list, setAll: bool = True, correctMisspelling: bool = False)
    Bases: geography.locator.Locator
    Adds context information to a place name
```

getRegions(`countryName: str`) → list
 get a list of regions for the given `countryName`
 `countryName(str)`: the `countryName` to check

get_region_names(`countryName: str`) → list
 get region names for the given country

Parameters `countryName` (*str*) – the name of the country

```
setAll()
    Set all context information
```

```
set_cities()
    set the cities information
```

```
set_countries()
    get the country information from my places
```

```
set_other()
```

```
set_regions()
    get the region information from my places (limited to the already identified countries)
```

1.6 geography.prefixtree module

1.7 geography.utils module

```
class geography.utils.Download
Bases: object

Utility functions for downloading data

static downloadBackupFile(url: str, fileName: str, targetDirectory: str, force: bool = False)
Downloads from the given url the zip-file and extracts the file corresponding to the given fileName.

Parameters
• url – url linking to a downloadable gzip file
• fileName – Name of the file that should be extracted from gzip file
• targetDirectory (str) – download the file this directory
• force (bool) – True if the download should be forced

Returns Name of the extracted file with path to the backup directory

static getFileContent(path: str)

static getURLContent(url: str)

static needsDownload(filePath: str, force: bool = False) → bool
check if a download of the given filePath is necessary that is the file does not exist has a size of zero or the
download should be forced

Parameters
• filePath (str) – the path of the file to be checked
• force (bool) – True if the result should be forced to True

Returns True if a download for this file needed

Return type bool

class geography.utils.Profiler(msg, profile=True)
Bases: object

simple profiler

time(extraMsg="")
time the action and print if profile is active

geography.utils.fuzzy_match(s1, s2, max_dist=0.8)
Fuzzy match the given two strings with the given maximum distance jellyfish jaro_winkler_similarity based on
https://en.wikipedia.org/wiki/Jaro-Winkler\_distance :param s1: string: First string :param s2: string: Second
string :param max_dist: float: The distance - default: 0.8

Returns True if the match is greater equals max_dist. Otherwise false

geography.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 geography.wikidata module

Created on 2020-09-23

@author: wf

```
class geography.wikidata.Wikidata(endpoint='https://query.wikidata.org/sparql', profile: bool = True)
Bases: object
Wikidata access

getCities(limit=1000000)
    get all human settlements as list of dict with duplicates for label, region, country ...

getCitiesForRegion(regionId, msg)
    get the cities for the given Region

getCityStates(limit=None)
    get city states from Wikidata
    try query

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(limit=None)
    get a list of countries
    try query

getRegions(limit=None)
    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

query(msg, queryString: str, limit=None) → list
    get the query result
    Parameters
        • msg (str) – the profile message to display
        • queryString (str) – the query to execute
    Returns the list of dicts with the result
```

Return type list

store2DB (*lod, tableName: str, primaryKey: str = None, sqlDB=None*)
store the given list of dicts to the database

Parameters

- **lod** (*list*) – the list of dicts
- **tableName** (*str*) – the table name to use
- **primaryKey** (*str*) – primary key (if any)
- **sqlDB** (*SQLDB*) – target SQL database

1.9 Module contents

main geography 3 module

geography.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

geography.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

geography.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

CHAPTER 3

tests package

3.1 Submodules

3.2 tests.test_extractor module

```
class tests.test_extractor.TestExtractor(methodName='runTest')
Bases: tests.basetest.Geography3Test
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
testExtractorFromText()
    test different texts for getting geo context information
testExtractorFromUrl()
    test the extractor
testGeographyIssue32()
    test https://github.com/ushahidi/geography/issues/32
testGetGeoPlace()
    test geo place handling
testIssue10()
    test https://github.com/somnathrakshit/geography3/issues/10 Add ISO country code
testIssue7()
    test https://github.com/somnathrakshit/geography3/issues/7 disambiguating countries
```

```
testIssue9()
test https://github.com/somnathrakshit/geography3/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-geographypython

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

testStackoverflow54077973()
see https://stackoverflow.com/questions/54077973/geography3-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-geograph

testStackoverflow55548116()
see https://stackoverflow.com/questions/55548116/geography3-library-is-not-working-properly-and-give-traceback-error

testStackoverflow62152428()
see https://stackoverflow.com/questions/62152428/extracting-country-information-from-description-using-geography?
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: tests.basetest.Geography3Test

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

checkExpected(lod, expected)

lookupQuery(viewName, whereClause)

testCityLookup()
    test the cityLookup to city/region/country object cluster

testCountryLookup()
    test country Lookup

testDelimiters()
    test the delimiter statistics for names

testExamples()
    test examples

testGetCountry()
    test getting a country by name or ISO

testHasViews()
    test that the views are available

testIsoRegexp()
    test regular expression for iso codes
```

```

testIssue15()
https://github.com/somnathrakshit/geography3/issues/15 test Issue 15 Disambiguate via population, gdp
data

testIssue17()
test issue 17:

https://github.com/somnathrakshit/geography3/issues/17
[BUG] San Francisco, USA and Auckland, New Zealand should be locatable #17

testIssue19()
test issue 19

testIssue22()
https://github.com/somnathrakshit/geography3/issues/22

testIssue41_CountriesFromErdem()
test getting Country list from Erdem

testIssue_42_distance()
test haversine and location

testIssue_59_db_download()
tests the correct downloading of the backup database in different configurations

testProceedingsExample()
test a proceedings title Example

testRegionLookup()
test region Lookup

testStackOverflow64379688()
compare old and new geography interface

testStackOverflow64418919()
https://stackoverflow.com/questions/64418919/problem-retrieving-region-in-us-with-geography3

testUML()
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: tests.basetest.Geography3Test

test Places

setUp()
setUp test environment

testGetRegionNames()
test getting region names

testIssue25()
https://github.com/somnathrakshit/geography3/issues/25

testIssue49()
country recognition

```

```
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: tests.basetest.Geography3Test  
  
    test the wikidata access for cities  
  
    testGetCoordinateComponents()  
        test the splitting of coordinate components in WikiData query results  
  
    testGetWikidataId()  
        test getting a wikiDataId from a given URL  
  
    testWikidataCities()  
        test getting city information from wikidata  
  
    testWikidataCityStates()  
        test getting region information from wikidata  
  
    testWikidataCountries()  
        test getting country information from wikidata  
  
    testWikidataRegions()  
        test getting region information from wikidata
```

3.7 Module contents

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

geography, 12
geography.extraction, 1
geography.labels, 1
geography.locator, 2
geography.places, 9
geography.utils, 10
geography.wikidata, 11

t

tests, 18
tests.test_extractor, 15
tests.test_locator, 16
tests.test_places, 17
tests.test_wikidata, 18

Index

A

add() (*geography.locator.LocationManager method*), 5

B

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

C

check() (*tests.test_extractor.TestExtractor method*), 15
checkExamples() (*tests.test_locator.TestLocator method*), 16
checkExpected() (*tests.test_locator.TestLocator method*), 16
cities (*geography.locator.LocationContext attribute*), 4
cities_for_name() (*geography.locator.Locator method*), 6
City (*class in geography.locator*), 2
CityManager (*class in geography.locator*), 2
correct_country_misspelling() (*geography.locator.Locator method*), 6
countries (*geography.locator.LocationContext attribute*), 4
Country (*class in geography.locator*), 3
country (*geography.locator.City attribute*), 2
country (*geography.locator.Region attribute*), 9
CountryManager (*class in geography.locator*), 3
createViews() (*geography.locator.Locator method*), 6

D

db_filename (*geography.locator.LocationContext attribute*), 4
db_has_data() (*geography.locator.Locator method*), 6
db_recordCount() (*geography.locator.Locator method*), 6
default (*geography.labels.Labels attribute*), 2
disambiguate() (*geography.locator.Locator method*), 7
distance() (*geography.locator.Location method*), 3
Download (*class in geography.utils*), 10

downloadBackupFile() (*geography.utils.Download static method*), 10

downloadBackupFileFromGitHub() (*geography.locator.LocationManager class method*), 5

downloadDB() (*geography.locator.Locator method*), 7

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
fromCache() (*geography.locator.LocationContext class method*), 4
fromCache() (*geography.locator.LocationManager method*), 5
fromCityLookup() (*geography.locator.City static method*), 2
fromCountryLookup() (*geography.locator.Country static method*), 3
fromErdem() (*geography.locator.CountryManager class method*), 3
fromRecord() (*geography.locator.Location class method*), 3
fromRegionLookup() (*geography.locator.Region static method*), 9
fuzzy_match() (*in module geography.utils*), 10

G

geo (*geography.labels.Labels attribute*), 2
geography (*module*), 12
geography_extraction (*module*), 1
geography_labels (*module*), 1
geography_locator (*module*), 2
geography_places (*module*), 9

```
geography.utils (module), 10
geography.wikidata (module), 11
get_geoPlace_context () (in module geography), 12
get_place_context () (in module geography), 12
get_region_names () (geography.places.PlaceContext method), 9
getAliases () (geography.locator.Locator method), 7
getBackupDirectory () (geography.locator.LocationManager method), 5
getBallTuple () (geography.locator.LocationManager method), 5
getByName () (geography.locator.LocationManager method), 6
getCities () (geography.wikidata.Wikidata method), 11
getCitiesForRegion () (geography.wikidata.Wikidata method), 11
getCityStates () (geography.wikidata.Wikidata method), 11
getCoordinateComponents () (geography.wikidata.Wikidata static method), 11
getCountries () (geography.wikidata.Wikidata method), 11
getCountry () (geography.locator.Locator method), 7
getConfig () (geography.locator.LocationContext static method), 5
getFileContent () (geography.utils.Download static method), 10
getInstance () (geography.locator.Locator static method), 7
getJSONFiles () (geography.locator.CityManager class method), 2
getLocationByID () (geography.locator.LocationManager method), 6
getLocationByIsoCode () (geography.locator.LocationManager method), 6
getLocationsByWikidataId () (geography.locator.LocationManager method), 6
getLocationsWithinRadius () (geography.locator.Location method), 4
getNClosestLocations () (geography.locator.Location method), 4
getRegions () (geography.places.PlaceContext method), 9
getRegions () (geography.wikidata.Wikidata method), 11
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), 9
getURLContent () (geography.utils.Download static method), 10
getValuesClause () (geography.wikidata.Wikidata static method), 11
getView () (geography.locator.Locator method), 7
getWikidataId () (geography.wikidata.Wikidata static method), 11
```

H

```
haversine () (geography.locator.Location static method), 4
```

I

```
interlinkLocations () (geography.locator.LocationContext method), 5
is_a_country () (geography.locator.Locator method), 7
isISO () (geography.locator.Locator static method), 7
isKnownAs () (geography.locator.Location method), 4
```

L

```
Labels (class in geography.labels), 2
load () (geography.locator.LocationContext method), 5
loadDB () (geography.locator.Locator method), 7
locateCity () (geography.locator.Locator method), 8
locateCity () (in module geography), 12
locateLocation () (geography.locator.LocationContext method), 5
Location (class in geography.locator), 3
LocationContext (class in geography.locator), 4
LocationManager (class in geography.locator), 5
Locator (class in geography.locator), 6
locator (geography.locator.Locator attribute), 8
lookupQuery () (tests.test_locator.TestLocator method), 16
```

M

```
main () (in module geography.locator), 9
mappedDict () (geography.locator.Location static method), 4
```

N

```
needsDownload () (geography.utils.Download static method), 10
```

normalizePlaces ()	(geography.locator.Locator static method), 8	split () (geography.extraction.Extractor method), 1
		store2DB () (geography.wikidata.Wikidata method), 12
P		T
partialDict ()	(geography.locator.Location static method), 4	testCityLookup () (tests.test_locator.TestLocator method), 16
PlaceContext (class in geography.places), 9		testCountryLookup () (tests.test_locator.TestLocator method), 16
places_by_name ()	(geography.locator.Locator method), 8	testDelimiters () (tests.test_locator.TestLocator method), 16
populate_Cities ()	(geography.locator.Locator method), 8	testExamples () (tests.test_locator.TestLocator method), 16
populate_Countries ()	(geography.locator.Locator method), 8	TestExtractor (class in tests.test_extractor), 15
populate_db ()	(geography.locator.Locator method), 8	testExtractorFromText ()
populate_Regions ()	(geography.locator.Locator method), 8	(tests.test_extractor.TestExtractor method), 15
populate_Version ()	(geography.locator.Locator method), 8	testExtractorFromUrl () (tests.test_extractor.TestExtractor method), 15
Profiler (class in geography.utils), 10		testGeographyIssue32 () (tests.test_extractor.TestExtractor method), 15
Q		testGetCoordinateComponents () (tests.test_wikidata.TestWikidata method), 18
query ()	(geography.wikidata.Wikidata method), 11	testGetCountry () (tests.test_locator.TestLocator method), 16
R		testGetGeoPlace () (tests.test_extractor.TestExtractor method), 15
radius (geography.locator.Earth attribute), 3		testGetRegionNames () (tests.test_places.TestPlaces method), 17
readCSV ()	(geography.locator.Locator method), 8	testGetWikidataId () (tests.test_wikidata.TestWikidata method), 18
recreateDatabase ()	(geography.locator.Locator method), 8	testHasViews () (tests.test_locator.TestLocator method), 16
Region (class in geography.locator), 9		testIsoRegexp () (tests.test_locator.TestLocator method), 16
region (geography.locator.City attribute), 2		testIssue10 () (tests.test_extractor.TestExtractor method), 15
RegionManager (class in geography.locator), 9		testIssue15 () (tests.test_locator.TestLocator method), 16
regions (geography.locator.LocationContext attribute), 5		testIssue17 () (tests.test_locator.TestLocator method), 17
regions_for_name ()	(geography.locator.Locator method), 8	testIssue19 () (tests.test_locator.TestLocator method), 17
remove_non_ascii ()	(in module geography.utils), 10	testIssue22 () (tests.test_locator.TestLocator method), 17
resetInstance ()	(geography.locator.Locator static method), 9	testIssue25 () (tests.test_places.TestPlaces method), 17
S		testIssue41_CountriesFromErdem () (tests.test_locator.TestLocator method), 17
set_cities ()	(geography.places.PlaceContext method), 9	
set_countries ()	(geography.places.PlaceContext method), 9	
set_other ()	(geography.places.PlaceContext method), 9	
set_regions ()	(geography.places.PlaceContext method), 9	
set_text ()	(geography.extraction.Extractor method), 1	
setAll ()	(geography.places.PlaceContext method), 9	
setUp ()	(tests.test_places.TestPlaces method), 17	
setValue ()	(geography.locator.City method), 2	

testIssue49() (*tests.test_places.TestPlaces method*), 17
testIssue7() (*tests.test_extractor.TestExtractor method*), 15
testIssue9() (*tests.test_extractor.TestExtractor method*), 15
testIssue_42_distance() (*tests.test_locator.TestLocator method*), 17
testIssue_59_db_download() (*tests.test_locator.TestLocator method*), 17
TestLocator (*class in tests.test_locator*), 16
TestPlaces (*class in tests.test_places*), 17
testPlaces() (*tests.test_places.TestPlaces method*), 17
testProceedingsExample() (*tests.test_locator.TestLocator method*), 17
testRegionLookup() (*tests.test_locator.TestLocator method*), 17
tests (*module*), 18
tests.test_extractor (*module*), 15
tests.test_locator (*module*), 16
tests.test_places (*module*), 17
tests.test_wikidata (*module*), 18
testStackoverflow43322567() (*tests.test_extractor.TestExtractor method*), 16
testStackoverflow54077973() (*tests.test_extractor.TestExtractor method*), 16
testStackoverflow54712198() (*tests.test_extractor.TestExtractor method*), 16
testStackOverflow54721435() (*tests.test_extractor.TestExtractor method*), 16
testStackoverflow55548116() (*tests.test_extractor.TestExtractor method*), 16
testStackoverflow62152428() (*tests.test_extractor.TestExtractor method*), 16
testStackOverflow64379688() (*tests.test_locator.TestLocator method*), 17
testStackOverflow64418919() (*tests.test_locator.TestLocator method*), 17
testUML() (*tests.test_locator.TestLocator method*), 17
TestWikidata (*class in tests.test_wikidata*), 18
testWikidataCities() (*tests.test_wikidata.TestWikidata method*), 18
testWikidataCityStates() (*tests.test_wikidata.TestWikidata method*), 18
testWikidataCountries()

testWikidataRegions() (*tests.test_wikidata.TestWikidata method*), 18
testWordCount() (*tests.test_locator.TestLocator method*), 17
time() (*geography.utils.Profiler method*), 10

WWikidata (*class in geography.wikidata*), 11