NeoBase Documentation

Release 0.1

Alex PRENGERE

Sep 22, 2023

Contents

GeoBase. Rebooted.

```
>>> b = NeoBase()
>>> b.get('ORY', 'city_code_list')
['PAR']
>>> b.get('ORY', 'city_name_list')
['Paris']
>>> b.get('ORY', 'country_code')
'FR'
>>> b.distance('ORY', 'CDG')
34.87...
>>> b.get_location('ORY')
LatLng(lat=48.72..., lng=2.35...)
```

class neobase.neobase.**NeoBase**(*rows=None*, *date=None*, *duplicates=None*)

Bases: object

Main structure, a wrapper around a dict, with dict-like behavior.

DUPLICATES = True

FIELDS = (('iata_code', 0, None), ('name', 6, None), ('lat', 8, None), ('lng', 9, None

KEY = 0

__init__(*rows=None*, *date=None*, *duplicates=None*)

Initialize self. See help(type(self)) for accurate signature.

distance(*key_0*, *key_1*, *default=<object object>*)

Compute distance between two elements.

This is just a wrapper between the original haversine function, but it is probably the most used feature :)

Parameters

- **key_0** – the first key
- **key_1** – the second key

Returns the distance (km)

```
>>> b = NeoBase()
>>> b.distance('ORY', 'CDG')
34.87...
```

static distance_between_locations(*l0*, *l1*)

Great circle distance

Parameters

- **l0** – the LatLng tuple of the first location
- **l1** – the LatLng tuple of the second location

Returns the distance in kilometers

```
>>> NeoBase.distance_between_locations((48.84, 2.367), (43.70, 7.26)) # Paris␣
↪-> Nice
683.85...
```

Case of unknown location.

```
>>> NeoBase.distance_between_locations(None, (43.70, 7.26)) # returns None
```

find_closest_from(*key*, *N=1*, *from_keys=None*)

Same as find_closest_from_location, except the location is given not by a lat/lng, but with its key, like ORY or SFO. We just look up in the base to retrieve lat/lng, and call find_closest_from_location.

Parameters

- **key** – the key of the location
- **N** – the N closest results wanted
- **from_keys** – if None, it takes all keys in consideration, else takes from_keys iterable of keys to perform find_closest_from_location. This is useful to combine searches

Returns an iterable of (dist, key)

```
>>> b = NeoBase()
>>> list(b.find_closest_from('NCE'))
[(0.0, 'NCE')]
>>> list(b.find_closest_from('NCE', N=3))
[(0.0, 'NCE'), (5.07..., 'XCG@1'), (5.45..., 'XCG')]
```

find_closest_from_location(*lat_lng*, *N=1*, *from_keys=None*)

Concept close to find_near_location, but here we do not look for the keys radius-close to a location, we look for the closest key from this location, given by latitude/longitude.

Parameters

- **lat_lng** – the lat_lng of the location
- **N** – the N closest results wanted
- **from_keys** – if None, it takes all keys in consideration, else takes from_keys iterable of keys to perform find_closest_from_location. This is useful to combine searches

Returns an iterable of (dist, key)

```
>>> b = NeoBase()
>>> list(b.find_closest_from_location((43.70, 7.26))) # Nice
[(0.60..., 'NCE@1')]
>>> list(b.find_closest_from_location((43.70, 7.26), N=3)) # Nice
[(0.60..., 'NCE@1'), (5.82..., 'NCE'), (5.89..., 'XBM')]
```

find_near(*key*, *radius=50*, *from_keys=None*)

Same as find_near_location, except the location is given not by a lat/lng, but with its key, like ORY or SFO. We just look up in the base to retrieve lat/lng, and call find_near_location.

Parameters

- **key** – the key of the location
- **radius** – the radius of the search (kilometers)
- **from_keys** – if None, it takes all keys in consideration, else takes from_keys iterable of keys to perform search.

Returns an iterable of (dist, key)

```
>>> b = NeoBase()
>>> sorted(b.find_near('ORY', 10)) # Orly, por <= 10km
[(0.0, 'ORY'), (6.94..., 'XJY'), (9.96..., 'QFC')]
```

find_near_location(*lat_lng*, *radius=50*, *from_keys=None*)

Returns a list of nearby keys from a location (given latidude and longitude), and a radius for the search.

Note that the haversine function, which compute distance at the surface of a sphere, here returns kilometers, so the radius should be in kms.

Parameters

- **lat_lng** – the lat_lng of the location
- **radius** – the radius of the search (kilometers)
- **from_keys** – if None, it takes all keys in consideration, else takes from_keys iterable of keys to perform search.

Returns an iterable of (dist, key)

```
>>> b = NeoBase()
>>> # Paris, airports <= 50km
>>> [b.get(k, 'iata_code') for d, k in sorted(b.find_near_location((48.84, 2.
↪367), 5))]
['PAR', 'XGB', 'XHP', 'XPG', 'XEX']
```

find_with(*conditions*, *from_keys=None*, *reverse=False*)

Get iterator of all keys with particular field.

For example, if you want to know all airports in Paris.

Parameters

- **conditions** – a list of (field, value) conditions
- **reverse** – we look keys where the field is *not* the particular value

Returns an iterator of matching keys

Testing several conditions.

```
>>> b = NeoBase()
>>> c0 = [('city_code_list', ['PAR'])]
>>> c1 = [('location_type', ['H'])]
>>> len(list(b.find_with(c0)))
16
>>> len(list(b.find_with(c0 + c1)))
2
```

get(*key*, *field=None*, *default=<object object>*)

Get data from structure.

```
>>> b = NeoBase()
>>> b.get('OR', 'city_code_list', default=None)
>>> b.get('ORY', 'city_code_list')
['PAR']
>>> b.get('nce', 'city_code_list')
['NCE']
```

get_location(*key*, *default=<object object>*)

Get None or the geocode.

```
>>> b = NeoBase()
>>> b.get_location('ORY')
LatLng(lat=48.72..., lng=2.35...)
```

keys()

Returns iterator of all keys in the base.

Returns the iterator of all keys

```
>>> b = NeoBase()
>>> sorted(b.keys())
['AAA', 'AAA@1', 'AAB', ...
```

classmethod load(*f*, *date*, *duplicates*)

Building a dictionary of geographical data from optd_por.

```
>>> import os.path as op
>>> path = op.join(op.dirname(__file__), 'optd_por_public.csv')
>>> with open_(path) as f:
...     b = NeoBase.load(f, '2030-01-01', True)
>>> b['ORY']['city_code_list']
['PAR']
```

set(*key*, ***data*)

Set information.

```
>>> b = NeoBase()
>>> b.get('ORY', 'name')
'Paris Orly Airport'
>>> b.set('ORY', name='test')
>>> b.get('ORY', 'name')
'test'
>>> b.set('Wow!', name='test')
>>> b.get('Wow!', 'name')
'test'
```

static skip(*row*, *date*)

class neobase.neobase.**LatLng**(*lat*, *lng*)

Bases: tuple

lat

Alias for field number 0

lng

Alias for field number 1

exception neobase.neobase.**UnknownKeyError**

Bases: KeyError

Python Module Index

n

Index

Symbols

D

F

G

K

L

N

S

U