geo-utils

Release 0.0.2

Sebastian Schwindt, Beatriz Negreiros, Kilian Mouris

Nov 16, 2020

CONTENTS

Geospatial Utility Functions for Hydraulics and Morphodynamics

`geo_utils` provides *Python3* functions for many sorts of river-related analyses with geospatial data. The package is intended as support material for the lecture Python programming for Water Resources Engineering and Research, where primarily *conda* environments are used. Even though `geo_utils` basically works in all *Python3* environments, make sure to follow the *Get Started* instructions on hydro-informatics.github.io to get ready with *Anaconda* and familiarize with git.

Note: This documentation is also as available as style-adapted PDF (download).

CHAPTER

ONE

INSTALLATION

Use `git` to download the `geo_utils` repository (make sure to install Git Bash):

1. Open *Git Bash* (or any other git-able *Terminal*)
2. Create or select a target directory for `geo_utils` (e.g., in your *Python* project folder)
3. Type `cd "D:/Target/Directory/"` to change to the target installation directory.
4. Clone the repository.

```
cd "D:/Target/Directory/"
git clone https://github.com/hydro-informatics/geo-utils.git
```

Now, `geo_utils` lives in `"D:/Target/Directory/geo-utils/geo_utils"`.

CHAPTER

TWO

USAGE

2.1 Import

1. Run *Python* and add the download directory of `geo-utils` to the system path:

```
import os, sys
sys.path.append("D:/Target/Directory/geo-utils/")  # Of course: replace "D:/Target/
↪Directory/", e.g., with  r'' + os.path.abspath('')
```

2. Import `geo_utils`:

```
import geo_utils.geo_utils as gu
```

2.2 Example

```
import geo_utils as gu
raster, array, geo_transform = gu.raster2array("/sample-data/froude.tif")
type(raster)
# >>> <class 'osgeo.gdal.Dataset'>
type(array)
# >>> <class 'numpy.ndarray'>
type(geo_transform)
# >>> <class 'tuple'>
print(geo_transform)
# >>> (6748604.7742, 3.0, 0.0, 2207317.1771, 0.0, -3.0)
```

CHAPTER

THREE

REQUIREMENTS

- Python 3.x (read more on hydro-informatics.github.io)
- Dependencies:
  - alphashape
  - fiona
  - gdal (read more on hydro-informatics.github.io/geo-pckg)
  - geojson
  - geopandas
  - numpy
  - pandas
  - pyshp
  - shapely

CHAPTER

FOUR

CODE STRUCTURE

The following diagram highlights function locations in *Python* scripts and how those are linked to each other.

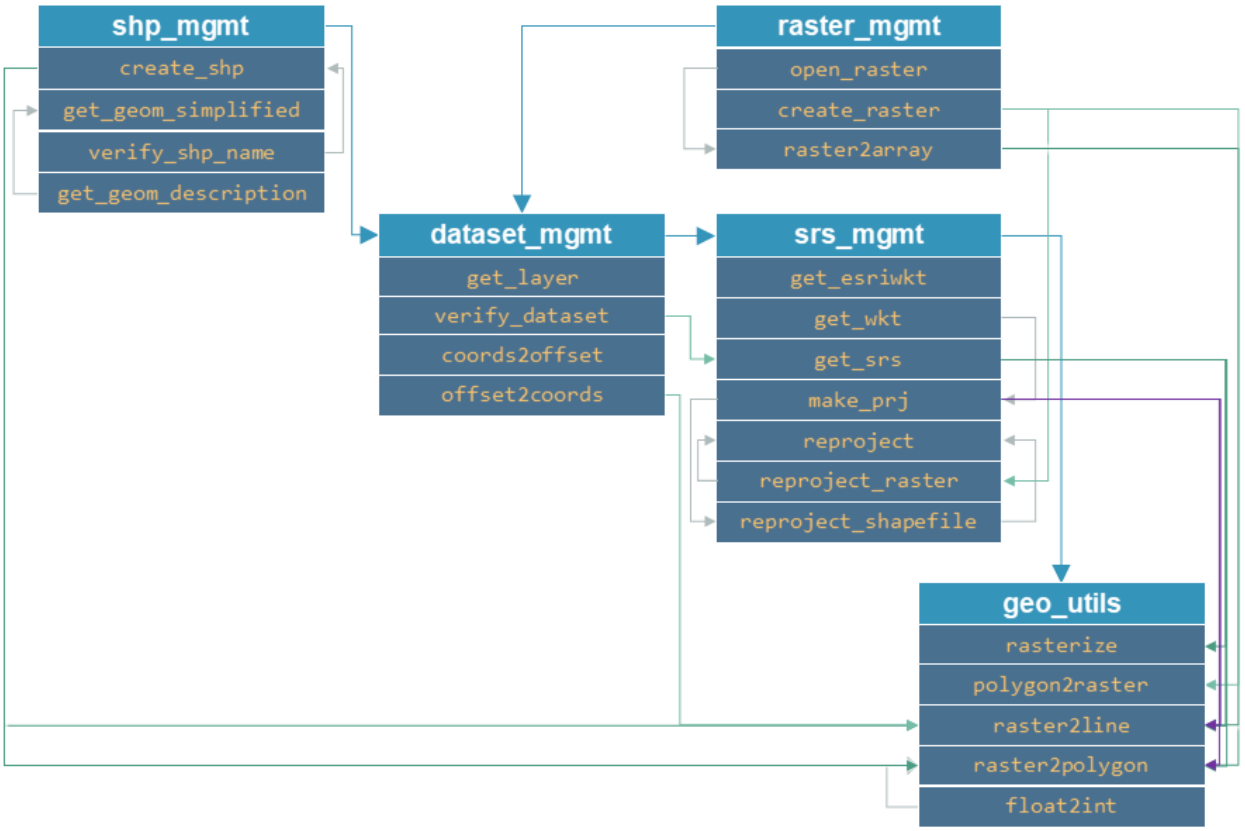

Fig. 1: *Diagram of the code structure (needs to be updated).*

CHAPTER

FIVE

SCRIPT AND FUNCTION DOCS

5.1 geo_utils (MASTER)

geo_utils is a package for creating, modifying, and transforming geo-spatial datasets. A detailed documentation of geo_utils is available at geo-utils.readthedocs.io.

geo_utils.geo_utils.**float2int**(*raster_file_name*, *band_number=1*)

Converts a float number raster to an integer raster (required for converting a raster to a polygon shapefile).

Parameters

- **raster_file_name** (*str*) – Target file name, including directory; must end on ".tif".
- **band_number** (*int*) – The raster band number to open (default: 1).

Returns "path/to/ew_raster_file.tif"

Return type str

geo_utils.geo_utils.**raster2line**(*raster_file_name*, *out_shp_fn*, *pixel_value*)

Converts a raster to a line shapefile, where pixel_value determines line start and end points.

Parameters

- **raster_file_name** (*str*) – of input raster file name, including directory; must end on ".tif".
- **out_shp_fn** (*str*) – of target shapefile name, including directory; must end on ".shp".
- **pixel_value** – Pixel values to connect.

geo_utils.geo_utils.**raster2polygon**(*file_name*, *out_shp_fn*, *band_number=1*, *field_name='values'*)

Converts a raster to a polygon shapefile.

Parameters

- **file_name** (*str*) – Target file name, including directory; must end on ".tif"
- **out_shp_fn** (*str*) – Shapefile name (with directory e.g., "C:/temp/poly.shp")
- **band_number** (*int*) – Raster band number to open (default: 1)
- **field_name** (*str*) – Field name where raster pixel values will be stored (default: "values")
- **add_area** – If True, an "area" field will be added, where the area in the shapefiles unit system is calculated (default: False)

geo_utils.geo_utils.**rasterize**(*in_shp_file_name*, *out_raster_file_name*, *pixel_size=10*, *no_data_value=-9999*, *rdtype=gdal.GDT_Float32*, *overwrite=True*, *interpolate_gap_pixels=False*, ***kwargs*)

Converts any ESRI shapefile to a raster.

Parameters

- **in_shp_file_name** (*str*) – A shapefile name (with directory e.g., `"C:/temp/poly.shp"`)
- **out_raster_file_name** (*str*) – Target file name, including directory; must end on `".tif"`
- **pixel_size** (*float*) – Pixel size as multiple of length units defined in the spatial reference (default: `10`)
- **no_data_value** (*int OR float*) – Numeric value for no-data pixels (default: `-9999`)
- **rdtype** (*gdal.GDALDataType*) – The raster data type (default: `gdal.GDT_Float32` (32 bit floating point)
- **overwrite** (*bool*) – Overwrite existing files (default: `True`)
- **interpolate_gap_pixels** (*bool*) – Fill empty pixels that are not touched by a shapefile element with interpolated values (default: `False`)

Keyword Arguments

- **field_name** (*str*) – Name of the shapefile's field with values to burn to raster pixel values.
- **radius1** (*float*) – Define the x-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.
- **radius2** (*float*) – Define the y-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.
- **power** (*float*) – Power of the function for interpolating pixel values (default: `1.0`, corresponding to linear).
- **smoothing** (*float*) – Smoothing parameter for interpolating pixel values (default: `0.0`).
- **min_points** (*int*) – Minimum number of points to use for interpolation. If the interpolator cannot find at least `min_points` for a pixel, it assigns a `no_data` value to that pixel (default: `0`).
- **max_points** (*int*) – Maximum number of points to use for interpolation. The interpolator will not use more than `max_points` closest points to interpolate a pixel value (default: `0`).

Hints: More information on pixel value interpolation: * `interpolate_gap_pixels=True` interpolates values at pixels that are not touched by any las point. * The pixel value interpolation uses `gdal_grid` (i.e., its Python bindings through `gdal.Grid()`). * Control the interpolation parameters with the keyword arguments `radius1`, `radius2`, `power`, `max_points`, `min_points`, and `smoothing`..

Returns Creates the GeoTIFF raster defined with `out_raster_file_name` (success: `0`, otherwise `None`).

Return type int

5.2 raster_mgmt raster management

geo_utils.raster_mgmt.**clip_raster**(*polygon*, *in_raster*, *out_raster*)

Clips a raster to a polygon.

Parameters

- **polygon** (*str*) – A polygon shapefile name, including directory; must end on ".shp".
- **in_raster** (*str*) – Name of the raster to be clipped, including its directory.
- **out_raster** (*str*) – Name of the target raster, including its directory.

Returns Creates a new, clipped raster defined with out_raster.

Return type None

geo_utils.raster_mgmt.**create_raster**(*file_name*, *raster_array*, *bands=1*, *origin=None*, *epsg=4326*, *pixel_width=10.0*, *pixel_height=10.0*, *nan_val=-9999.0*, *rdtype=gdal.GDT_Float32*, *geo_info=False*, *rotation_angle=None*, *shear_pixels=True*, *options=['PROFILE=GeoTIFF']*)

Converts an ndarray (numpy.array) to a GeoTIFF raster.

Parameters

- **file_name** (*str*) – Target file name, including directory; must end on ".tif".
- **raster_array** (ndarray or list) – Array or list of arrays of values to rasterize. If a list of arrays is provided, the length of the list will correspond to the number of bands added to the raster (supersedes bands).
- **bands** (*int*) – Number of bands to write to the raster (default: 1).
- **origin** (*tuple*) – Coordinates (x, y) of the origin.
- **epsg** (*int*) – EPSG:XXXX projection to use (default: 4326).
- **pixel_height** (*float OR int*) – Pixel height as multiple of the base units defined with the EPSG number (default: 10 meters).
- **pixel_width** (*float OR int*) – Pixel width as multiple of the base units defined with the EPSG number (default: 10 meters).
- **nan_val** (int or float) – No-data value to be used in the raster. Replaces non-numeric and np.nan in the ndarray. (default: geoconfig.nan_value).
- **rdtype** – gdal.GDALDataType raster data type (default: gdal.GDT_Float32 (32 bit floating point).
- **geo_info** (*tuple*) – Defines a gdal.DataSet.GetGeoTransform object and supersedes origin, pixel_width, pixel_height (default: False).
- **rotation_angle** (*float*) – Rotate (in degrees) not North-up rasters. The default value (0) corresponds to north-up (only modify if you know what you are doing).
- **shear_pixels** (*bool*) – Use with rotation_angle to shear pixels as well (default: True).
- **options** (*list*) – Raster creation options - default is ['PROFILE=GeoTIFF']. Add 'PHOTOMETRIC=RGB' to create an RGB image raster.

Returns 0 if successful, otherwise -1.

Return type int

Hint: For processing airborne imagery, the `roation_angle` corresponds to the bearing angle of the aircraft with reference to true, not magnetic North.

`geo_utils.raster_mgmt.`**`open_raster`**(*file_name*, *band_number=1*)
Opens a raster file and accesses its bands.

Parameters

- **`file_name`** (*`str`*) – The raster file directory and name.
- **`band_number`** (*`int`*) – The Raster band number to open (default: `1`).

Returns A raster dataset a Python object. osgeo.gdal.Band: The defined raster band as Python object.

Return type osgeo.gdal.Dataset

`geo_utils.raster_mgmt.`**`raster2array`**(*file_name*, *band_number=1*)
Extracts an `ndarray` from a raster.

Parameters

- **`file_name`** (*`str`*) – Target file name, including directory; must end on `".tif"`.
- **`band_number`** (*`int`*) – The raster band number to open (default: `1`).

Returns Indicated raster band, where no-data values are replaced with `np.nan`. GeoTransform: The GeoTransformation used in the original raster.

Return type ndarray

`geo_utils.raster_mgmt.`**`remove_tif`**(*file_name*)
Removes a GeoTIFF and its dependent files (e.g., xml).

Parameters **`file_name`** (*`str`*) – Directory (path) and name of a GeoTIFF

Returns Removes the provided `file_name` and all dependencies.

5.3 `shp_mgmt` shapefile management

`geo_utils.shp_mgmt.`**`create_shp`**(*shp_file_dir*, *overwrite=True*, **args*, ***kwargs*)
Creates a new shapefile with an optionally defined geometry type.

Parameters

- **`shp_file_dir`** (*`str`*) – of the (relative) shapefile directory (ends on `".shp"`).
- **`overwrite`** (*`bool`*) – If `True` (default), existing files are overwritten.
- **`layer_name`** (*`str`*) – The layer name to be created. If `None`: no layer will be created.
- **`layer_type`** (*`str`*) – Either `"point"`, `"line"`, or `"polygon"` of the `layer_name`. If `None`: no layer will be created.

Returns An `ogr` shapefile

Return type osgeo.ogr.DataSource

`geo_utils.shp_mgmt.`**`get_geom_description`**(*layer*)
Gets the WKB Geometry Type as string from a shapefile layer.

Parameters **`layer`** (*`osgeo.ogr.Layer`*) – A shapefile layer.

Returns WKB (binary) geometry type

Return type str

geo_utils.shp_mgmt.**get_geom_simplified**(*layer*)

Gets a simplified geometry description (either point, line, or polygon) as a function of the WKB Geometry Type of a shapefile layer.

Parameters **layer** (*osgeo.ogr.Layer*) – A shapefile layer.

Returns Either WKT-formatted point, line, or polygon (or unknown if invalid layer).

Return type str

geo_utils.shp_mgmt.**polygon_from_shapepoints**(*shapepoints*, *polygon*, *alpha=numpy.nan*)

Creates a polygon around a cloud of `shapepoints`.

Parameters

- **shapepoints** (*str*) – Point shapefile name, including its directory.
- **polygon** (*str*) – Target shapefile filename, including its directory.
- **alpha** (*float*) – Coefficient to adjust; the lower it is, the more slim will be the polygon.

Returns Creates the polygon shapefile defined with `polygon`.

Return type None

geo_utils.shp_mgmt.**verify_shp_name**(*shp_file_name*, *shorten_to=13*)

Ensure that the shapefile name does not exceed 13 characters. Otherwise, the function shortens the `shp_file_name` length to N characters.

Parameters

- **shp_file_name** (*str*) – A shapefile name (with directory e.g., `"C:/temp/poly.shp"`).
- **shorten_to** (*int*) – The number of characters the shapefile name should have (default: `13`).

Returns A shapefile name (including path if provided) with a length of `shorten_to`.

Return type str

5.4 `srs_mgmt` projection management

geo_utils.srs_mgmt.**get_esriwkt**(*epsg*)

Gets esriwkt-formatted spatial references with epsg code online.

Parameters **epsg** (*int*) – EPSG Authority Code

Returns An esriwkt string (if an error occur, the default epsg=``4326`` is used).

Return type str

Example

```
get_esriwkt(4326)
```

geo_utils.srs_mgmt.**get_srs**(*dataset*)
Gets the spatial reference of any gdal.Dataset.

Parameters **dataset** (*gdal.Dataset*) – A shapefile or raster.

Returns A spatial reference object.

Return type osr.SpatialReference

geo_utils.srs_mgmt.**get_wkt**(*epsg*, *wkt_format='esriwkt'*)
Gets WKT-formatted projection information for an epsg code using the osr library.

Parameters

- **epsg** (*int*) – epsg Authority code
- **wkt_format** (*str*) – of wkt format (default is esriwkt for shapefile projections)

Returns WKT (if error: returns default corresponding to epsg=4326).

Return type str

geo_utils.srs_mgmt.**make_prj**(*shp_file_name*, *epsg*)
Generates a projection file for a shapefile.

Parameters

- **shp_file_name** (*str*) – of a shapefile name (with directory e.g., "C:/temp/poly.shp").
- **epsg** (*int*) – EPSG Authority Code

Returns Creates a projection file (.prj) in the same directory and with the same name of shp_file_name.

geo_utils.srs_mgmt.**reproject**(*source_dataset*, *new_projection_dataset*)
Re-projects a dataset (raster or shapefile) onto the spatial reference system of a (shapefile or raster) layer.

Parameters

- **source_dataset** (*gdal.Dataset*) – Shapefile or raster.
- **new_projection_dataset** (*gdal.Dataset*) – Shapefile or raster with new projection info.

Returns

- If the source is a raster, the function creates a GeoTIFF in same directory as source_dataset with a "_reprojected" suffix in the file name.
- If the source is a shapefile, the function creates a shapefile in same directory as source_dataset with a "_reprojected" suffix in the file name.

geo_utils.srs_mgmt.**reproject_raster**(*source_dataset*, *source_srs*, *target_srs*)
Re-projects a raster dataset. This function is called by the reproject function.

Parameters

- **source_dataset** (*osgeo.ogr.DataSource*) – Instantiates with an ogr.Open(SHP-FILE).
- **source_srs** (*osgeo.osr.SpatialReference*) – Instantiates with get_srs(source_dataset)

- **target_srs** (*osgeo.osr.SpatialReference*) – Instantiates with `get_srs(DATASET-WITH-TARGET-PROJECTION)`.

Returns Creates a new GeoTIFF raster in the same directory where `source_dataset` lives.

`geo_utils.srs_mgmt.`**reproject_shapefile**(*source_dataset*, *source_layer*, *source_srs*, *target_srs*)

Re-projects a shapefile dataset. This function is called by the `reproject` function.

Parameters

- **source_dataset** (*osgeo.ogr.DataSource*) – Instantiates with `ogr.Open(SHP-FILE)`.
- **source_layer** (*osgeo.ogr.Layer*) – Instantiates with `source_dataset.GetLayer()`.
- **source_srs** (*osgeo.osr.SpatialReference*) – Instantiates with `get_srs(source_dataset)`.
- **target_srs** (*osgeo.osr.SpatialReference*) – Instantiates with `get_srs(DATASET-WITH-TARGET-PROJECTION)`.

Returns Creates a new shapefile in the same directory where `source_dataset` lives.

5.5 `dataset_mgmt` dataset conversion

`geo_utils.dataset_mgmt.`**coords2offset**(*geo_transform*, *x_coord*, *y_coord*)

Returns x-y pixel offset (inverse of the `offset2coords` function).

Parameters

- **geo_transform** – osgeo.gdal.Dataset.GetGeoTransform() object
- **x_coord** (*float*) – x-coordinate
- **y_coord** (*float*) – y-coordinate

Returns Number of pixels `(offset_x, offset_y)`, both `int` .

Return type tuple

`geo_utils.dataset_mgmt.`**get_layer**(*dataset*, *band_number=1*)

Gets a `layer=band` (`RasterDataSet`) or `layer=ogr.Dataset.Layer` of any dataset.

Parameters

- **dataset** (`osgeo.gdal.Dataset` or `osgeo.ogr.DataSource`) – Either a raster or a shapefile.
- **band_number** (*int*) – Only use with rasters to define a band number to open (default=``1``).

Returns `{GEO-TYPE: if raster: raster_band, if vector: GetLayer(), else: None}`

Return type dict

`geo_utils.dataset_mgmt.`**offset2coords**(*geo_transform*, *offset_x*, *offset_y*)

Returns x-y coordinates from pixel offset (inverse of `coords2offset` function).

Parameters

- **geo_transform** (*osgeo.gdal.Dataset.GetGeoTransform*) – The geo transformation to use.
- **offset_x** (*int*) – x number of pixels.
- **offset_y** (*int*) – y number of pixels.

Returns Two `float` numbers of x-y-coordinates `(x_coord, y_coord)`.

Return type tuple

`geo_utils.dataset_mgmt.`**verify_dataset**(*dataset*)

Verifies if a dataset contains raster or vector data.

Parameters **dataset** (`osgeo.gdal.Dataset` or `osgeo.ogr.DataSource`) – Dataset to verify.

Returns Either "mixed", "raster", or "vector".

Return type string

5.6 KML/KML file management

Modified script (original: Linwood Creekmore III)

Examples

output to geopandas dataframe (gdf) gdf = kmx2other("my-places.kmz", output="gpd")

plot the new gdf (use %matplotlib inline in notebooks) gdf.plot()

convert a kml-file to a shapefile success = kmx2other("my-places.kml", output="shp")

`geo_utils.kml.`**kmx2other**(*file*, *output='df'*)

Converts a Keyhole Markup Language Zipped (KMZ) or KML file to a pandas dataframe, geopandas geodataframe, csv, geojson, or ESRI shapefile.

Parameters

- **file** (*str*) – The path to a KMZ or KML file.
- **output** (*str*) – Defines the output type. Valid options are: `"shapefile"`, `"shp"`, `"shapefile"`, or `"ESRI Shapefile"`.

Hint: The core function is taken from http://programmingadvent.blogspot.com/2013/06/kmzkml-file-parsing-with-python.html

---

Returns object

Return type self

Original Classes written by Linwood Creekmore III (modified for geo_utils)

With code blocks from:

- http://programmingadvent.blogspot.com/2013/06/kmzkml-file-parsing-with-python.html
- http://gis.stackexchange.com/questions/159681/geopandas-cant-save-geojson
- https://gist.github.com/mciantyre/32ff2c2d5cd9515c1ee7

class geo_utils.kmx_parser.**ModHTMLParser**
A child of HTMLParser, tailored (modified) for kml/kmy parsing.

handle_data(*data*)
Generates mapping and series if `in_table` is `True`.

Parameters **data** (*str*) – Text lines of data divided by colons.

handle_starttag(*tag*, *attrs*)
Enables a table if a table-tag is provided.

Parameters

- **tag** (*str*) – Set to "table" for enabling usage of a table.
- **attrs** (*list*) – List of additional attributes (currently unused).

class geo_utils.kmx_parser.**PlacemarkHandler**
Child of `xml.sax.handler.ContentHandler`, tailored for handling kml files.

characters(*data*)
Adds a line of data to the read-buffer.

Parameters **data** (*str*) –

endElement(*name*)
Sets the end (last) element.

Parameters **name** (*str*) –

htmlizer()
Creates an html file.

spatializer()
Converts string objects to spatial Python objects.

Parameters **row** (*df*) – List of strings for conversion

startElement(*name*, *attributes*)
Looks for the first Placemark element in a kml file.

Parameters

- **name** (*str*) – Name-tag of the element
- **attributes** (*str*) –

CHAPTER

SIX

EXAMPLES

An example application for projecting non-georeferenced TIF files on a spatial reference system. The TIF files in this example have a prefix and suffix name, which are separated by a 4-digits number. The origin of the new GeoTIFF files is derived from a KML file.

CHAPTER

SEVEN

CONTRIBUTING

7.1 How to document

This package uses *Sphinx* readthedocs and the documentation regenerates automatically after pushing changes to the repositories `main` branch.

To set styles, configure or add extensions to the documentation use `ROOT/.readthedocs.yml` and `ROOT/docs/conf.py`.

Functions and classes are automatically parsed for docstrings and implemented in the documentation. `hylas` docs use google style docstring formats - please familiarize with the style format and strictly apply in all commits.

To modify this documentation file, edit `ROOT/docs/index.rst` (uses reStructuredText format).

In the class or function docstrings use the following section headers:

- `Args (alias of Parameters)`
- `Arguments (alias of Parameters)`
- `Attention`
- `Attributes`
- `Caution`
- `Danger`
- `Error`
- `Example`
- `Examples`
- `Hint`
- `Important`
- `Keyword Args (alias of Keyword Arguments)`
- `Keyword Arguments`
- `Methods`
- `Note`
- `Notes`
- `Other Parameters`
- `Parameters`
- `Return (alias of Returns)`

- Returns
- Raise (alias of Raises)
- Raises
- References
- See Also
- Tip
- Todo
- Warning
- Warnings (alias of Warning)
- Warn (alias of Warns)
- Warns
- Yield (alias of Yields)
- Yields

For local builds of the documentation, the following packages are required:

```
$ sudo apt-get install build-essential
$ sudo apt-get install python-dev python-pip python-setuptools
$ sudo apt-get install libxml2-dev libxslt1-dev zlib1g-dev
$ apt-cache search libffi
$ sudo apt-get install -y libffi-dev
$ sudo apt-get install python3-dev default-libmysqlclient-dev
$ sudo apt-get install python3-dev
$ sudo apt-get install redis-server
```

To generate a local html version of the `hylas` documentation, `cd` into the `docs` directory and type:

```
make html
```

Learn more about *Sphinx* documentation and the automatic generation of *Python* code docs through docstrings in the tutorial provided at github.com/sschwindt/docs-with-sphinx

CHAPTER

EIGHT

INDICES AND TABLES

- :ref:`genindex`
- :ref:`modindex`
- :ref:`search`

PYTHON MODULE INDEX

g

INDEX

S

V