Bearing / Azimuth Documentation

Release 0.1.0

Keith Sanders

Apr 15, 2021

Contents:

CHAPTER 1

Bearing / Azimuth Converter

Simple application to convert between bearings and azimuth. This application came about when I found my simple CAD program did not accept bearings as input. So, the best choice was to build a little converter so that values could be copied to the clipboard and pasted into the CAD program.

- Free software: MIT license
- Documentation: https://bearingazimuth.readthedocs.io/en/latest/
- Github Repository: https://github.com/Shakiestnerd/BearingAzimuth

1.1 Features

- Enter a bearing in the form N 45° 30’ 00” E and have it converted to an azimuth (angle from north).
- Enter the azimuth angle and have the bearing automatically calculated.
- Copy the results to the clipboard
- Draw a sample angle on screen.
- User interface created with PySimpleGui

1.2 Credits

Used the Anaconda distribution for development.

PySimpleGui is the name of my virtual environment.

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install bearing, run this command in your terminal:

```
$ pip install bearing
```

This is the preferred method to install bearing, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for bearing can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/shakiestnerd/bearing
```

Or download the tarball:

```
$ curl -OJL https://github.com/shakiestnerd/bearing/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

The Bearing / Azimuth application is a stand-alone utility that does a conversion between bearings and azimuth. As you enter the information into the form, the fields will update based on your input.

Note: The Bearing class located in angle.py. This class may be useful outside of this little utility.

CHAPTER 4

bearing

4.1 bearing package

4.1.1 Submodules

4.1.2 bearing.angle module

This module includes the Bearing class which handles all the bearings and azimuth values and performs all the conversions.

class bearing.angle.**Bearing**

Bases: object

The Bearing class handles storing the data for bearing and azimuth. It does the conversion between the two and returns the information in several formats.

calc_azimuth() → None

Performs the azimuth conversion using the private members

Returns None

calc_bearing(*az: float*) → None

Performs the bearing calculation. :param az: Azimuth as a float

Returns None

dec_to_dms() → None

Adjust the angle based on north and sets the degrees, minutes, and seconds member variables.

Returns None

get_azimuth() → float

Since accessing the member variables directly is discouraged, this method returns the azimuth value.

Returns azimuth value

Return type float

get_bearing() → str
Return just the bearing as a formatted string.

Returns bearing value

Return type str

get_bearing_dict() → Dict[KT, VT]
Return the components of a bearing in a dictionary.

Returns bearing

Return type Dict

set_azimuth(*az: float*) → str
Set the value of the azimuth and perform the bearing conversion.

Parameters **az** (*float*) – azimuth value

Returns The bearing as a string

Return type str

set_bearing(*n: str*, *d: int*, *m: int*, *s: int*, *e: str*) → float
Initialize a bearing and perform the azimuth conversion

Parameters

- **n** (*str*) – northing (is always either 'N' or 'S')
- **d** (*int*) – degrees
- **m** (*int*) – minutes
- **s** (*int*) – seconds
- **e** (*str*) – easting (always either 'E' or 'W')

Returns the azimuth as a float

Return type float

submit_azimuth(*az: str*) → Dict[KT, VT]
Sets the azimuth value when the input is a string. Does some validation and returns the bearing as a dictionary.

Parameters **az** (*str*) – azimuth value

Returns bearing components

Return type Dict

submit_bearing(*n: str*, *d: str*, *m: str*, *s: str*, *e: str*) → float
Initialize a bearing using string values for input. submit_bearing does some extra validation and converts degrees minutes, and seconds to integers.

Parameters

- **n** (*str*) – northing (is always either 'N' or 'S')
- **d** (*str*) – degrees
- **m** (*str*) – minutes
- **s** (*str*) – seconds
- **e** (*str*) – easting (always either 'E' or 'W')

Returns the azimuth as a float or 0 on failure

Return type float

4.1.3 bearing.bearing module

Bearing main module. Runs the user interface to allow entering the bearing and azimuth values

class bearing.bearing.**UI**
Bases: object

The UI class defines the user interface dialog using PySimpleGUI. It allows entering the bearing and azimuth values and performs the conversions as text is entered into the form fields.

draw_axis()
Draws the X and Y axis on the canvas for the graphical representation of the angle on the user interface form.

draw_vector()
Draws the angle on the canvas. Adds a little arc to illustrate.

4.1.4 Module contents

Top-level package for bearing.

CHAPTER 5

Contributing

What follows is the boilerplate contributing guidelines that were included with Audrey's python cookie cutter package. I like them and it is goal I aspire to be able to support. So, for that reason, I am leaving the information in this file.

This is a small project that allows me to get comfortable with developing and Deploying a project along with testing and documentation that should be part of a complete package.

At this time, I only feel comfortable with looking at reported bugs.

Thank you for your understanding.

CHAPTER 6

Future Contribution Goals

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/shakiestnerd/bearing/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

6.1.4 Write Documentation

bearing could always use more documentation, whether as part of the official bearing docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/shakiestnerd/bearing/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *bearing* for local development.

1. Fork the *bearing* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/bearing.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv bearing
$ cd bearing/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 bearing tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/shakiestnerd/bearing/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ pytest tests.test_bearing
```

6.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 7

Credits

7.1 Development Lead

- Keith Sanders <keithmo@canofworms.com>

7.2 Contributors

None yet. Why not be the first?

CHAPTER 8

History

8.1 0.1.0 (2020-04-19)

- First release on PyPI.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index