GeoMath Documentation

Release 0.1.2

Vinicius Mesel

February 08, 2017

Contents

GeoMath is a Mathematical Library for Python 3.X that tries to solve every problem included in the Analytical Geometry field, you can use in any of your apps for getting several calculus done in miliseconds.

The library is Object Oriented and it's very simple to use(as we used the KISS - Keep it Simple, Stupid - Standards), to use the library you need some instances like `Point` or `Line` or even a `Figure`, that will help you saving time and getting things done quickly.

A single script can solve as many problems as you want, but keep in mind that you will do this scripts, but the library will give all the resources needed to do it.

The project is named *GeoMath* as a tribute to all the mathematicians that worked with the geometry field, during this milleniums.

CHAPTER 1

Installation

There are two simple ways to get GeoMath installed and running on your computer:

1. from Python PyPi
2. from github(not recommended)

1.1 Installing from PyPi

To install GeoMath from PyPi, first, you have to make sure that you have `pip` installed in your computer. With it installed, you just need to run:

```
` pip install geomath `
```

If everything goes fine with your installation, you just installed GeoMath and you can use it in Python.

1.2 Installing from Github

Since GeoMath is under beta stages and during a very hard development state, installing GeoMath from it should not be a great idea. If you want to get the most stable version of the library you should install it via `pip`.

To install the library from Github, you must run this command:

```
` $ git clone https://www.github.com/vmesel/GeoMath.git `
```

After that, you should move GeoMath to the libraries folder in Python, so you can import it from any system.

Done! Now you can use GeoMath!

CHAPTER 2

More documentation

Contents:

2.1 Figures

The figures functions are made to handle figures made of points and calculate stuff like barycenter, points, Perimeter and others that we are working on!

2.1.1 Creating a Figure

`figure.`**`Figure`**`()`

This is the function to start creating figures. This function is very simple to be used, you just need to type in this with a variable to make the variable become the figure!

```
>>> FIGUREVAR = figure.Figure()
# This will create your figure
```

`figure.`**`addPoint`**`(`*Point*`)`

Figures are currently made of points, so if you need to make a figure have one more point, you will need to run addPoint

```
>>> FIGUREVAR.addPoint(point.Point(x,y))
```

The point that is added must be a GeoMath object, if it's not, it won't be recognized by the system.

`figure.`**`addPoints`**`(`[*Point1*, *Point2*, ...]`)`

If you want to add more than one point to an existing figure, you just need to run the command below. It will add the points to the assigned figure.

```
>>> FIGUREVAR.addPoints([point.Point(x,y), point.Point(x,y), ...])
```

The points that are added must be GeoMath objects, if they are not, they won't be recognized by the system.

2.1.2 Figure Attributes

`figure.`**`Barycenter`**`()`

This function is very useful, if you need to calculate the barycenter of a figure, it can be done with just one line of code like this:

```
>>> FIGUREVAR.Barycenter()
# This will return your figure barycenter point
```

2.1.3 Functions Under Development

`figure.`**`Perimeter`**`()`

This function gets the perimeter of the figure that you have created with the points.

```
>>> FIGUREVAR.Perimeter()
# This will return your figure perimeter
```

`figure.`**`Area`**`()`

This function calculates the area of the figure that you have created with the points.

```
>>> FIGUREVAR.Area()
# This will return your figure perimeter
```

2.2 Lines

The line statement is the basic statement for the Euclidean Geometry and for everything in geometry. Figures are formed by lines, distances are the length of lines and even more. Lines must be used to get general arguments, like angles, line equation, comparisons between to lines and etc.

2.2.1 Creating the Line

To start with a line you may want to run these commands that are below this statement.

`line.`**`Line`**`()`

To start getting a line done in the GeoMath system, you must run this command! This is an object assigned function, so you must assign it to an object so it can be called

```
>>> LINEVAR = line.Line()
```

There are two ways of inputting values for a line: creating a line with 2 Points or by the Line Equation(supposing that you have it).

`line.`**`create`**`(`*PointOne*, *PointTwo*`)`

To get a line assigning two different points to it, you must run this command. It will generate a line in the instance that is in your line.

```
>>> LINEVAR.create(point.Point(0,0), point.Point(4,4))
```

Running these commands, we will generate the equation of the line simply by solving the matrix represented by the two points and by the x and y line.

`line.`**`create_via_equation`**`(`*LineEquation*`)`

These command is very useful if you have the line equation. The line equation is inserted by a string formated like this “ax-by+c=0”: with the coefficients in the front and with the line

```
>>>  LINEVAR.create_via_equation(LineEquation)
```

2.2.2 Line Attributes

In this section, we will explain to you the attributes that the Line() class have in the project!

`line.`**`equation`**`()`

This is a commmand that will enable you to get the equation of the line you have created.

```
>>> LINEVAR.Equation()
```

`line.`**`get_angle`**`()`

This function gets the angulation of your line and give it in degrees

```
>>> LINEVAR.get_angle()
```

`line.`**`A`**`()`

This function returns the A coefficient.

```
>>> LINEVAR.A()
```

`line.`**`B`**`()`

This function returns the B coefficient.

```
>>> LINEVAR.B()
```

`line.`**`C`**`()`

This function returns the C coefficient.

```
>>> LINEVAR.C()
```

2.2.3 Attributes involving other functions

There are some attributed that depend on other functions, and we are going to list them here below.

`line.`**`point_distance`**`(`*self*, *Point*`)`

This function is made to calculate the distance between a line and a point with a very simple syntax.

```
>>>  LINEVAR.point_distance(Point)
```

`line.`**`comparisor`**`(`*self*, *Line*`)`

This function was made to compare two lines and decide if they are parallel, coincident or perpendicular.

```
>>>  LINEVAR.comparisor(Line)
```

`line.`**`point_alignment`**`(`*self*, *Point*`)`

This function is to check the 3 point alignment, but for using it, you need to create a line and check the alignment of a point in the line.

```
>>>  LINEVAR.point_alignment(Point)
```

2.3 Point

The point attributes are very simple, we have some of them that involves two points, but they are quite simple to use like we describe here below:

2.3.1 Setting Up a Point

`point.`**`Point`**(*x*, *y*)

This is the function to start creating points to make your life easier and happier. This function is very simple to be used, you just need to type in:

```
>>> PointOne = point.Point(x,y) # where x and y are the coordinates of your point
# This will return a point
```

2.3.2 Point attributes with other Points

`point.`**`midpoint`**(*OtherPoint*)

This is the function that you will want to use when you need to calculate the midpoint from two points, so to use it, you first instantiate the first point and them:

```
>>> MidPoint = POINTVariable.point.midpoint(OtherPoint)
# This will return the MidPoint between POINTVariable and OtherPoint
```

`point.`**`distance`**(*OtherPoint*)

If you need to calculate the distance between two points, you don't need to do any difficult calculations for that, you just need to call the function like this:

```
>>> Distance = POINTVariable.distance(OtherPoint)
# This will return the distance between POINTVariable and OtherPoint
```

2.3.3 Point Attributes

`point.`**`quadrant`**()

If you need to know in which quadrant your point is located at, we already solved this problem for you, you just need to use point.quadrant and it will return a int that represents the quadrant:

```
>>> Distance = POINTVariable.point.quadrant()
# This will return the quadrant from POINTVariable
```

2.4 Contributing to GeoMath

As we are an open source library, we want everyone to interact with our content and suggest things to improve in our system. While the library is up and running, you will see updates and everything that can help you to develop more complex system solving apps. But to continue to improve, we need your help! Our initial development team is working on things for the library and for other projects, so if you want to help, here is a guide that you must follow to contribute.

2.4.1 Contribution Guideline

Everything you put to work on this library is valid as a contribution, but for the commit get approve, it must follow some simple rules.

Coded Contributions:

- You must have documented the changes you have made in the docs, so we can see what you implemented and changed
- You must comment in your code, so it can be readable for everyone that is going to use

If you don't want to contribute with code, you can donate for us some money, translate the library for your language, or even create a logo, so we won't be an ugly library.

2.4.2 Contributions Needed

We are looking after for some types of contributors:

- Mathematicians or anyone who loves geometry
- Beta testers for new functions
- Anyone who have any knowledge building GUIs for Apps in Python
- Design for a Logo

2.5 Indices and tables

- genindex
- modindex
- search

F

L

P