PHY1112 Lab 4

Classes and Modules

January 30th, 2024

|  |  |  |  |
| --- | --- | --- | --- |
| Part | 1 | 2 | Total |
| Points | 12 | 20 | 32 |
| Score |  |  |  |

Objectives

1. Create your first custom class, the Vector2 class.
2. Learn how to import and use modules.
3. Use the math module to add specific functionality to the Vector2 class.

Part 1: Custom classes: Vector2 – not new to you (but new to Python)

1. (5 points) Create a file called “lab4.py”. Create a class called Vector2, which when initialized takes in the arguments, initial\_x and initial\_y, and assigns them to the data attributes x and y of an instance of that class.

1. (5 points) Add a \_\_str\_\_() special method to your Vector2 class that represents the vector as a comma and space separated list of x and y, for example, in the format [3, -4].
2. (2 points) Test your class and \_\_str\_\_() method by creating an instance of Vector2 with and . Print it to the terminal. Include a screenshot of what has been printed.

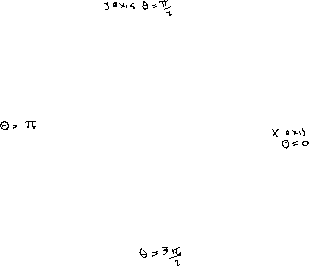
A screenshot of a computer

Description automatically generated

Part 2: Modules – Using other people’s code to improve our own (ethically)

The Vector2 objects as we have defined them above are in Cartesian coordinates, that is, they have an and a component.

Sometimes it is useful to write a vector in terms of its magnitude , and its angle from the axis. Vectors using the representation are said to be in polar coordinates. The and components have been labeled in the figure below for the vector represented by the blue solid arrow.



One can convert from Cartesian to polar coordinates using the following formulas:

1. (5 points) Add a method to Vector2 called magnitude()that calculates according to the formula given above. Make sure to use the math.sqrt() function.
2. (5 points) Add a method to Vector2 called angle\_from\_x\_axis() that calculates as according to the formula above. Use math.atan2(), that gives its output in radians within the range [-π,π]. You will need to convert this angle to be in the range [0,2π].
3. (5 points) Add a method to Vector2 called polar\_representation() that returns a tuple containing and by utilizing the methods you just created, Vector2.magnitude()and Vector2.angle\_from\_x\_axis().
4. (5 points) Create Vector2 objects with the following values, and print out their Cartesian, and polar representations. Write your output here and take a screenshot of what is printed by your program.

Note that part e) should throw an error, so make sure it is the last one you create after completely printing out everything from parts a) to d).

A computer screen shot of a number

Description automatically generated

**Remember to submit this document filled out, along with your python file on Brightspace!**

**CODE**

'''

Filename:       lab4.py

Author:         Patrick Geraghty

Date Created:   2023-01-30

Date Modified:  2023-01-30

Description:

'''

# Import math module for square root and pi per lab instructions. Import as m for brevity.

import math as m

*class* Vector2:

    # Initialize the class with the x and y coordinates

*def* \_\_init\_\_(*self*, *initial\_x*, *initial\_y*):

        self.x = initial\_x

        self.y = initial\_y

    # Define the string representation of the class

*def* \_\_str\_\_(*self*):

        return '[' + *str*(self.x) + ', ' + *str*(self.y) + ']'

    # Define a method to represent the polar magnitude of the class

*def* magnitude(*self*):

        return m.sqrt(self.x\*\*2 + self.y\*\*2)

    # Define a method to represent the polar angle of the class

*def* angle\_from\_x\_axis(*self*):

        if m.atan2(self.y, self.x) < 0:

            return m.atan2(self.y, self.x) + 2\*m.pi

        return m.atan2(self.y, self.x)

    # Define a method to represent the polar representation of the class

*def* polar\_representation(*self*):

        return [self.magnitude(), self.angle\_from\_x\_axis()]

# Define the test cases

a = Vector2(1, 0)

b = Vector2(3.2, 3.2)

c = Vector2(-1, -5.5)

d = Vector2(0, 0)

e = Vector2(3, 9*j*)

# Simple loop to print test cases, excluding error case

for i in [a, b, c, d]:

    print(i)

    print(*str*(i.polar\_representation()))

    print()

# Try to print the error case, and print the error message

try:

    print(e)

    print(*str*(e.polar\_representation()))

except *TypeError*:

    print('Error: Cannot convert complex number to polar representation')