Lab 07

AUTHOR
Christian Osendorfer

Implement in Python

Unless otherwise stated, don't use any modules that implement a solution to the questions asked. Come up with your doctests and compare these with those from other students.

Triangle

You are given the following class `Point`:

```
import math

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance_to_origin(self):
        return (self.x**2 + self.y**2)**0.5

    def euclidean_distance(self, other):
        return ((self.x - other.x)**2 + (self.y - other.y)**2)**0.5

    def manhattan_distance(self, other):
        return abs(self.x - other.x) + abs(self.y - other.y)

    def angle_between(self, other):
        vert = other.y - self.y
        horiz = other.x - self.x
        return math.atan2(vert, horiz)
```

Let's define a class `Triangle` to represent triangles in two dimensions.

The `__init__` method for triangles should take in three instances of `Point`, where each instance represents one vertex of the triangle. For example:

```
t = Triangle(Point(2,3), Point(5,6), Point(3, 5))
```

Your class must provide the following methods:

- A method `side_lengths()`, which returns a tuple containing the side lengths (in the same units as the coordinates), in any order.

    - Side lengths (and other quantities in this problem) are `float`s. `float`s can not be tested for equality. How can you work around that problem?
- A method `angles()`, which returns a list or tuple containing the angles present in the triangle (in radians), in any order.
- A method `side_classification()`, which returns a string: `"scalene"`, `"isosceles"`, or `"equilateral"`, depending on what type of triangle is represented.
- A method `angle_classification()`, which returns a string: `"acute"`, `"right"`, `"obtuse"`, or `"equiangular"`; whichever best describes the triangle.
- A method is_right(), which returns a boolean: True if this is a right triangle, and False otherwise.
- A method `area()`, which returns the area of the triangle.
- A method `perimeter()`, which returns the perimeter of the triangle.

Vector

Define a class called `Vector` to represent n-dimensional vectors.

The class's `__init__` method should take as an argument a list containing the numbers in the vector. Provide the following methods:

- a method `as_list()`, which returns a list containing the numbers in the vector.
- a method `size()`, which returns an integer containing the number of elements in the vector.
- a method `magnitude()`, which should return the magnitude of the vector.
- a method `euclidean_distance(other)`, which should return the Euclidean distance between self and other, where other is an instance of Vector with the same size.
- a method `normalized()`, which should return a unit vector (vector of length 1) pointing in the same direction as the original vector.
- a method `cosine_similarity(other)`, which should return the cosine of the angle between self and other, where self and other are vectors of the same size.
- a method `__add__(other)`, which returns a new instance of Vector representing the sum of the original instance and other:
    - if `other` *is an instance of* `Vector` with appropriate size, your code should perform a vector addition.
    - if `other` *is an instance of* `Vector` whose dimensions are inappropriate for vector addition, raise an appropriate error.
    - otherwise, raise an appropriate error!
- a method `__sub__(other)`, which should behave analogously to add, but should perform subtraction.

- a method `__mul__(other)`, which returns an object representing the result of multiplying the original instance and `other`: Implement the dot product (*Skalarprodukt*) between `self` and `other`, if everything is ok, otherwise, like before, raise appropriate errors!
- a method `__truediv__(other)`, which returns a new instance of Vector representing the result of dividing the original instance by `other`:
  - if `other` is an `int` or `float`, divide every entry by `other`.
  - otherwise, if `other` is an instance of `Vector` with the right dimension, divide each value of `self` by the respective value of `other`.
  - otherwise, raise an appropriate error(s).

Write a python program, using the `__main__` guard, in the same file as the class definition that showcases the class' functionality.

Compute without a computer

Consider this Python class:

```
class Car:
    color = 'gray'

    def describe_car(self):
        return 'A cool ' + Car.color + ' car'

    def describe_self(self):
        return 'A cool ' + self.color + ' car'
```

Provide the values that will be printed by the following program:

```
nona = Car()
print(nona.describe_car())

print(nona.describe_self())

print(Car.color)

print(nona.color)

lola = Car()
lola.color = 'plaid'
print(lola.describe_car())

print(lola.describe_self())

print(lola.color)

print(Car.color)

nona.size = 'small'
print(lola.size)
print(Car.size)

Car.size = 'big'
print(nona.size)

print(lola.size)
```

For strings, include the quotation marks, even though those would not actually be displayed on the screen. If a `print` call would cause an error, write `error`.