CECS 229 Programming Assignment #4

Due Date:

Sunday, 3/24 @ 11:59 PM

Submission Instructions:

Complete the programming problems in the file named `pa4.py` . You may test your implementation on your Repl.it workspace by running `main.py`. When you are satisfied with your implementation,

1. Submit your Repl.it workspace
2. Download the file `pa4.py` and submit it to the appropriate CodePost auto-grader folder.

Objectives:

1. Apply vector operations to translate, scale, and rotate a set of points representing an image.
2. Perform various operations with or on vectors: addition, subtraction, dot product, norm.

NOTES:

1. Unless otherwise stated in the FIXME comment, you may not change the outline of the algorithm provided by introducing new loops or conditionals, or by calling any built-in functions that perform the entire algorithm or replaces a part of the algorithm.
2. You may import and use the Python `math` module to obtain the value for e and to calculate sine, cosine, and tangent functions, if needed.

Problem 1:

Create a function `translate(S, z0)` that translates the points in the input set S by $z_0 = a_0 + b_0 i$. The function should satisfy the following:

1. INPUT:
   - `S` - set S
   - `z0` - complex number
2. OUT:
   - `T` - set T consisting of points in S translated by z_0

In [ ]:
```
def translate(S, z0):
    """
    translates the complex numbers of set S by z0
    :param S: set type; a set of complex numbers
    :param z0: complex type; a complex number
```

```
    :return: set type; a set consisting of points in S translated by z0
    """
    # FIXME: Implement this function
    # FIXME: Return correct output
    return None
```

Problem 2:

Create a function `scale(S, k)` that scales the points in the input set S by a factor of k:

1. INPUT:
   - `S` - set S
   - `k` - positive float, raises ValueError if k $\leq$ 0.
2. OUTPUT:
   - a set consisting of points in S scaled by k.

In [ ]:
```
def scale(S, k):
    """
    scales the complex numbers of set S by k.
    :param S: set type; a set of complex numbers
    :param k: float type; positive real number
    :return: set type; a set consisting of points in S scaled by k
    :raise: raises ValueError if k <= 0
    """
    # FIXME: Implement this function.
    # FIXME: Return correct output
    return None
```

Problem 3:

Create a function `rotate(S, tau)` that rotates the points in the input set S by τ radians:

1. INPUT:
   - `S` - set S
   - `tau` - float. If negative, the rotation is clockwise. If positive the rotation is counterclockwise. If zero, no rotation.
2. OUT:
   - a set consisting of points in S rotated by τ

In [ ]:
```
def rotate(S, tau):
    """
    rotates the complex numbers of set S by tau radians.
    :param S: set type; - set of complex numbers
    :param tau: float type; radian measure of the rotation value. If negative, the rot
                If positive the rotation is counterclockwise. If zero, no rotation.
    :returns: set type; a set consisting of points in S rotated by tau radians
    """
    # FIXME: Implement this function.
    # FIXME: Return correct output
    return None
```

Problem 4:

Finish the implementation of class `Vec` which instantiates row-vector objects with defined operations of addition, subtraction, scalar multiplication, and dot product. In addition, `Vec` class overloads the Python built-in function `abs()` so that when it is called on a `Vec` object, it returns the Euclidean norm of the vector.

In [ ]:
```
class Vec:
    def __init__(self, contents = []):
        """
        Constructor defaults to empty vector
        INPUT: list of elements to initialize a vector object, defaults to empty list
        """
        self.elements = contents
        return

    def __abs__(self):
        """
        Overloads the built-in function abs(v)
        :returns: float type; the Euclidean norm of vector v
        """
        # FIXME: Implement this method
        # FIXME: Return correct output
        return None

    def __add__(self, other):
        """
        overloads the + operator to support Vec + Vec
        :raises: ValueError if vectors are not same length
        :returns: Vec type; a Vec object that is the sum vector of this Vec and 'other
        """
        # FIXME: Finish the implementation
        # FIXME: Return correct output
        return None

    def __sub__(self, other):
        """
        overloads the - operator to support Vec - Vec
        :raises: ValueError if vectors are not same length
        :returns: Vec type; a Vec object that is the difference vector of this Vec and
        """
        # FIXME: Finish the implementation
        # FIXME: Return correct output
        return None



    def __mul__(self, other):
        """
        Overloads the * operator to support
            - Vec * Vec (dot product) raises ValueError if vectors are not
              same length in the case of dot product; returns scalar
            - Vec * float (component-wise product); returns Vec object
            - Vec * int (component-wise product); returns Vec object

        """
```

```
        if type(other) == Vec: #define dot product
            # FIXME: Complete the implementation
            # FIXME: Return the correct output
            return None

        elif type(other) == float or type(other) == int: #scalar-vector multiplicatior
            # FIXME: Complete the implementation
            # FIXME: Return the correct output
            return None


    def __rmul__(self, other):
        """Overloads the * operation to support
            - float * Vec; returns Vec object
            - int * Vec; returns Vec object
        """
        # FIXME: Complete the implementation
        # FIXME: Return the correct output
        return None



    def __str__(self):
        """returns string representation of this Vec object"""
        return str(self.elements) # does NOT need further implementation
```