Ex. No: 2 12-04-2023	UIT2201 — Programming and Data Structures

Aim:

To execute the following programs and note the output.

PART – A

1. Define a class Point, a simple class to represent 2-dimensional points (Non-mutable). Each object has two fields: '_x' and '_y'. Methods include 'distance' that returns Euclidean distance between 'this' object and another object.

Code:

```
"""
    This module provides functionality for finding the Euclidean distance
    between 2 Point objects.

    Original Author: Pranesh Kumar

    Created On: 12 Apr 2023
"""


class Point:
    def __init__(self, a: float = 0, b: float = 0):
        """Constructor of Point class

        Args:
            a (float, optional): x coordinate of Point. Defaults to 0.
            b (float, optional): y coordinate of Point. Defaults to 0.
        """
        self.__x = a
        self.__y = b
        self.__x_diff = 0
        self.__y_diff = 0
        self.__distance = 0

    def distance(self, anotherobject) -> float:
        """Find the Euclidean distance between 2 points

        The Euclidean distance between 2 points using the formula

        d = (((x1-x2)**2 + (y1-y2)**2)**0.5)

        Args:
            anotherobject (Point): another Point object to find the
distance

        Returns:
            float: distance between 2 objects
        """
```

```
        self.__x_diff = (self.__x - anotherobject.__x) ** 2
        self.__y_diff = (self.__y - anotherobject.__y) ** 2
        self.__distance = (self.__x_diff + self.__y_diff) ** 0.5
        return self.__distance

    def __str__(self) -> str:
        """Returns human-readable string representation of an object

        Returns:
            str: String representation of object
        """
        return f"({self.__x} , {self.__y})"


# driver code
if __name__ == "__main__":
    x1 = float(input("Enter x coordinate of first point:"))
    y1 = float(input("Enter y coordinate of first point:"))
    x2 = float(input("Enter x coordinate of second point:"))
    y2 = float(input("Enter y coordinate of second point:"))

    p1 = Point(x1, y1)
    p2 = Point(x2, y2)

    print("Point p1 is:", p1)
    print("Point p2 is:", p2)

    print("Distance between p1 and p2 is: ", p1.distance(p2))
    print("Distance between p2 and p1 is: ", p2.distance(p1))
```

Inputs and Output:

```
Enter x coordinate of first point:-12.75
Enter y coordinate of first point:66.45
Enter x coordinate of second point:0.25
Enter y coordinate of second point:16.778
Point p1 is: (-12.75 , 66.45)
Point p2 is: (0.25 , 16.778)
Distance between p1 and p2 is:  51.344985967473015
Distance between p2 and p1 is:  51.344985967473015
```

Explanation:

This is a Python module that provides a class called Point for finding the Euclidean distance between two points in a two-dimensional plane. The Point class has two attributes, x and y, which represent the coordinates of the point. The distance() method calculates the Euclidean distance between two Point objects using the formula ((x1-x2)**2 + (y1-y2)**2)**0.5. The module contains a driver code that prompts the user to enter the

coordinates of two points and creates Point objects based on the input. Finally, the driver code calculates and prints the distance between the two points using the distance() method.

Possible Test Cases:

Here are some possible test cases that could be used to test the Point class and its distance() method:

1. Test case where both Point objects are located at the same point. The expected output for the distance() method should be 0.

2. Test case where both Point objects are located on the x-axis. The expected output for the distance() method should be the absolute difference between their x-coordinates.

3. Test case where both Point objects are located on the y-axis. The expected output for the distance() method should be the absolute difference between their y-coordinates.

4. Test case where both Point objects are located on a diagonal line. The expected output for the distance() method should be the distance between the two points calculated using the Pythagorean theorem.

5. Test case where one or both Point objects have negative coordinates. The expected output for the distance() method should be the same as for the corresponding positive coordinates.

6. Test case where one or both Point objects have non-integer coordinates. The expected output for the distance() method should be the distance calculated with the floating-point values.

7. Test case where one or both of the arguments passed to the distance() method are not Point objects. The method should raise an appropriate exception or return an error message.

8. Test case where the attributes of the Point objects are accessed directly instead of using the distance() method. This should not be done, as the attributes are private and should not be accessed from outside the class.

2. Write a Python code to generate a random sequence of n Points. Define a function that, given an integer k and a new Point Pnew, returns k-nearest neighbours of Pnew in the given sequence of n Points.

Code:

```
"""
    This module provides functionality for finding the k-nearest neighbours
    of a given point with some k random points

    Original Author: Pranesh Kumar

    Created on : 12 Apr 2023
"""

# importing the necessary modules
from question1 import Point
import random


def findKNN(k: int, newPoint: Point, startvalue: int = 0, endvalue: int =
100) -> list:
    """This function finds the k-nearest neighbours of a given point with k
    random points generated.

    It returns the points which are closest to the newPoint in ascending
order

    Args:
        k (int): number of points to be generated
        newPoint (Point): new object of Point class whose KNN is to be
determined
        startvalue (int, optional): Starting range for generating random
values. Defaults to 0.
        endvalue (int, optional): Starting range for generating random
values. Defaults to 100.

    Returns:
        list: list of points in ascending order of distance from newPoint
    """
    distances = {}
    for _ in range(k):
        myPoint = Point(
            random.randint(startvalue, endvalue),
random.randint(startvalue, endvalue)
        )
        distances[str(myPoint)] = newPoint.distance(myPoint)

    distancesList = list(distances)
    knearestneighbours = sorted(distancesList, key=lambda x: x[1])

    return knearestneighbours


# driver code
if __name__ == "__main__":
    n = int(input("Enter the number of test cases: "))
    x1 = 12
```

```
y2 = 21
p = Point(x1, y2)
print(findKNN(n, p))
```

Output:

```
Enter the number of test cases: 5
['(18 , 81)', '(36 , 23)', '(66 , 83)', '(76 , 69)', '(77 , 19)']
```

Explanation:

This module defines a function named 'findKNN' that finds the k-nearest neighbors of a given point with k random points. The function takes in four parameters - k, newPoint, startvalue and endvalue, where k is an integer representing the number of points to be generated, newPoint is an object of the Point class whose KNN is to be determined, and startvalue and endvalue represent the range of values to generate random points. The function returns a list of points in ascending order of distance from the newPoint. The module also imports the 'Point' class from question1 module and uses it to create Point objects. The driver code takes the number of test cases as input, creates a Point object 'p', and calls the 'findKNN' function passing the number of test cases and the Point object as parameters, and then prints the result.

Test Cases:

For the second module , some possible test cases could be:

1. Generating KNN with k=0, which should return an empty list.

2. Generating KNN with k=1 and comparing the output to the original Point object.

3. Generating KNN with k > 1 and ensuring the list is sorted in ascending order of distance from the newPoint.

4. Testing the module's ability to handle invalid input, such as non-integer values for k or non-Point values for newPoint.

5. Testing the module's ability to handle edge cases, such as generating KNN with very large or very small values for startvalue and endvalue.

==========