**IMPLEMENTATION – EX 3 :**

**Question 1:**

import math

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def getPoint(self):

return self.x, self.y

def showPoint(self):

print(f"Point: ({self.x}, {self.y})")

class Shape(Point):

def \_\_init\_\_(self, x, y, vertices):

super().\_\_init\_\_(x, y)

self.vertices = vertices

def identifyShape(self):

num\_vertices = len(self.vertices)

if num\_vertices == 3:

return "Triangle"

elif num\_vertices == 4:

side\_lengths = []

for i in range(4):

x1, y1 = self.vertices[i]

x2, y2 = self.vertices[(i + 1) % 4]

side\_length = math.sqrt((x2 - x1) \*\* 2 + (y2 - y1) \*\* 2)

side\_lengths.append(side\_length)

# Assuming a square has equal sides

if all(side == side\_lengths[0] for side in side\_lengths):

return "Square"

# Assuming a rectangle has opposite sides of equal length

elif side\_lengths[0] == side\_lengths[2] and side\_lengths[1] == side\_lengths[3]:

return "Rectangle"

return "Unknown Shape"

class Outlier(Shape):

def checkIfPointInShape(self, x\_point, y\_point):

# Assuming the shape is a square with vertices at (self.x, self.y), (self.x + side\_length, self.y),

# (self.x, self.y + side\_length), and (self.x + side\_length, self.y + side\_length)

side\_length = math.sqrt((self.vertices[1][0] - self.vertices[0][0]) \*\* 2 + (self.vertices[1][1] - self.vertices[0][1]) \*\* 2)

if self.x <= x\_point <= self.x + side\_length and self.y <= y\_point <= self.y + side\_length:

print("Point falls within the area.")

else:

print("Point is an outlier.")

# Main method to demonstrate the classes

if \_\_name\_\_ == "\_\_main\_\_":

# Create a Point

point = Point(2, 3)

point.showPoint()

# Create a Shape (Assuming a square with vertices)

vertices = [(2, 3), (4, 3), (2, 5), (4, 5)]

shape = Shape(7, 9, vertices)

print("Identified Shape:", shape.identifyShape())

# Create an Outlier and check if a point falls within the shape

outlier\_point = Outlier(3, 4, vertices)

x\_point, y\_point = 3.5,4.5

print(f"Checking point ({x\_point}, {y\_point})")

outlier\_point.checkIfPointInShape(x\_point, y\_point)

**OUTPUT:**

Point: (2, 3)

Identified Shape: Rectangle

Checking point (3.5, 4.5)

Point falls within the area.

**Question 2:**

class Node:

'''creating a class node '''

def \_\_init\_\_(self, item = None, prev = None, next = None , parent = None) :

self.item = item

self.left = prev

self.right = next

self.parent = parent

class Tree() :

'''creating a tree data structure to perform operation like insert , search , delete and traverse operations'''

def \_\_init\_\_(self) :

self.root = None

self.size = 0

def left (self,pos) :

return pos.left

def right(self ,pos):

return pos.right

def addroot(self,item) : # creates the root of the tree

if self.root is not None :

raise ValueError("root exits")

root = Node(item)

self.size = 1

return root

def addleft(self,item,pos) : # add the left node to the node by creating a new node with item

if pos is None:

raise TypeError('Not a valid position.')

if self.left(pos) is not None :

raise ValueError("item is there")

else:

pos.left = Node(item,parent = pos)

self.size += 1

return pos.left

def addright(self,item,pos) : # add the right node to the node by creating a new node with item

if pos is None:

raise TypeError('Not a valid position.')

if self.right(pos) is not None :

raise ValueError("item is there")

else:

pos.right = Node(item,parent = pos)

self.size += 1

return pos.right

def parent(self,pos) :

return pos.parent

def insert(self,element,pos) : # insert the element in the tree

if pos == None:

self.root = self.addroot(element)

while pos is not None:

if pos.item > element :

if pos.left is None :

self.addleft(element,pos)

break

else :

return (self.insert(element, pos.left))

else :

if pos.right is None :

self.addright(element,pos)

break

else :

return (self.insert(element,pos.right))

def search(self,element,pos) : # search the elrement in the tree

if pos.item == element:

return True

elif pos.item > element :

return (self.search(element, pos.left))

elif pos.item < element :

return (self.search(element,pos.right))

else :

return False

def address(self,element,pos) : # search the elrement in the tree

if pos.item == element:

return pos

elif pos.item > element :

return (self.address(element, pos.left))

elif pos.item < element :

return (self.address(element,pos.right))

else :

return None

def findmin (self,pos) : # returns the mininum element of the tree

if pos.left is None :

return pos

else :

return self.findmin(pos.left)

def delete (self,element) :

pos = self.address(element,self.root)

Parent = self.parent(pos)

if pos.left == None and pos.right == None : # delete the node which has no child

if Parent.left == pos :

Parent.left = None

self.size -= 1

elif Parent.right == pos :

Parent.right = None

self.size -= 1

elif pos.left != None and pos.right == None : # delete the node which has left child alone

if Parent.left == pos :

Parent.left = pos.left

self.size -= 1

else:

Parent.right = pos.left

self.size -= 1

elif pos.left == None and pos.right != None : # delete the node which has right child alone

if Parent.left == pos :

Parent.left = pos.right

self.size -= 1

else:

Parent.right = pos.right

self.size -= 1

elif pos.left != None and pos.right != None

r = self.findmin(pos.right)

pos.item = r.item

r.item = 20000000

self.delete(r.item)

def traverse(self,pos):

if pos is None :

pos = self.root

if pos is not None :

if pos.left is not None :

self.traverse(pos.left)

print(pos.item)

if pos.right is not None :

self.traverse(pos.right)

a = Tree()

a.insert(6,a.root)

a.insert(5,a.root)

a.insert(8,a.root)

a.traverse(a.root)

print(a.search(5,a.root))

a.delete(6)

a.traverse(a.root)

from inheritanceTREES import Tree

import os

class FilesSystemTree(Tree):

'''This subclass simulate a file system hierarchy where each node represents a directory or file .'''

def search\_file (self, element ,pos) :

''' search for a file if it is present ,then return "true "or else return "false"

if the other than file is given as input ,then it return "it is not a file" '''

if os.path.isfile(element):

return super().search(element ,pos)

else:

print("It is not a file")

def add(self,element ,pos) : # add the files or directories in the tree

super().insert(element ,pos)

def display(self,x): # display the files or directories

super().traverse(x)

'''def delete(self,item): # delete the files or directories

super().delete(item)'''

if \_\_name\_\_ == '\_\_main\_\_' :

# creating a instance for child class

a = FilesSystemTree()

# add the files and directories

a.add("e:\it lab",a.root)

a.add("E:\it lab\SEM 3\Programming and Design Patterns\Lab",a.root)

a.add("E:\it lab\SEM 3\Programming and Design Patterns\Lab\library.py",a.root)

# displaying the files and directories

a.display(a.root)

# search a file

print(a.search\_file("E:\it lab\SEM 3\Programming and Design Patterns\Lab\library.py",a.root))

a.search\_file("E:\it lab\SEM 3\Programming and Design Patterns\Lab",a.root) # output : It is not a file

# delete a file or directory

a.delete("E:\it lab\SEM 3\Programming and Design Patterns\Lab")

# display the final list after a deletion

a.display(a.root)

from inheritanceTREES import Tree

class OrganisationTree (Tree):

'''This subclass simulate a organisational hierarchy where each node represents an employee or department .'''

def add(self,element ,pos) : # add the employees or departments in the tree

super().insert(element ,pos)

def search\_employee (self, element ,pos,position) :

''' search for a employee if he/she is present ,then return "true "or else return "false"

if the other than employee is given as input ,then it return "not an employee" '''

if position == "employee" :

return super().search(element ,pos)

else:

print("not an employee")

def display(self,x): # display the employees or department

super().traverse(x)

'''def delete(self,item): # delete the employees or department

super().delete(item)'''

if \_\_name\_\_ == '\_\_main\_\_' :

# creating a instance for child class

a = OrganisationTree()

# add the employees and departments

a.add("maths",a.root)

a.add("Ram",a.root)

a.add("IT",a.root)

# displaying the employees and departments

a.display(a.root)

# search an employee

print(a.search\_employee("Ram ",a.root,"employee"))

a.search\_employee("maths",a.root,"department")

# delete an employee or departments

a.delete("IT")

# display the final list after a deletion

a.display(a.root)

**Output:**

**IT**

**Ram**

**maths**

**True**

**not an employee**

**Ram**

**Maths**