In [ ]:
```
### Template
#import math for euclidean equations
import math

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

    def __str__(self):
        """ Returns a string with the format (x,y) """
        # -- YOUR CODE HERE --
        str_output=f"({self.x},{self.y})"
        return(str_output)

    def __eq__(self, other):
        """ Checks if two points pertain to the same coordinates """
        # -- YOUR CODE HERE --
        if(self.x==other.x and self.y==other.y):
            return(True)
        else:
            return(False)



    def get_distance(self, other):
        """ Calculates the distance between two points """
        # -- YOUR CODE HERE --
        disx=(self.x-other.x)^2
        disy=(self.y-other.y)^2
        dis_output=math.sqrt(disx+disy)
        return(dis_output)



class Line:
    def __init__(self, point_a, point_b):
        self.points = [point_a, point_b]


    def __str__(self):
        """ Returns a string containing all collinear points in the Line instance """
        # -- YOUR CODE HERE --

        #declare empty string
        str_point="["

        #process every point
        for i in self.points:
            #turn every point into a point class
            PrintPoint=Point(i[0],i[1])

            #get the string object
            print_holder=PrintPoint.__str__()

            #concatenate string to the final list
            str_point=str_point+print_holder+", "

        #remove the extra comma and space at the end
        str_point=str_point[:-2]+"]"
        return(str_point)


    def get_slope_intercept(self):
        """ Calculates the slope of the line """
        # -- YOUR CODE HERE --

        #attach the points onto variables
        Point_A=self.points[0]
        Point_B=self.points[1]

        #declare point objects
        PointObject_A=Point(int(Point_A[0]),int(Point_A[1]))
        PointObject_B=Point(int(Point_B[0]),int(Point_B[1]))

        #solve for slope using x and y attributes in the Point Oblects
        slope=(PointObject_B.y-PointObject_A.y)/(PointObject_B.x-PointObject_A.x)
        return(slope)

    def add_points(self, points):
        """ Adds a list of collinear points to the Line instance """
        # -- YOUR CODE HERE --

        #solve for the slope
        #to be used to check colinearity
        slope=self.get_slope_intercept()
```

```
        for i in points:

            #integer variable will be enabled only if input is not a duplicate or doesnt have an infinite slope
            allow=True

            #verify if the point is a duplicate or has an infinite slope
            for j in self.points:
                if(i==j or i[0]==j[0]):
                    allow=False
                    break

            #solve for slope only if allow is enabled
            #reason: infinite slope causes a divide by zero error
            if(allow):
                #goal: measure the slope between the input points and the self.point
                Compare_Line=Line(self.points[0],i)

                #use the Line Class function to solve for slope
                compare_slope=Compare_Line.get_slope_intercept()

                #finally compare the two slopes
                if(slope==compare_slope):
                    #only append the points to the list if they are valid
                    self.points.append(i)


    def remove_points(self, points):
        """ Removes a list of points from the Line instance """
        # -- YOUR CODE HERE --

        #Solution: use __eq__() function to verify if input point exists in self.points list
        for i in points:
            for j in self.points:
                allow=1
                #add if-case that prevents removal of initialized points
                if(j==self.points[0] or j==self.points[1]):
                    allow=0


                #create point objects
                PointObject_A=Point(int(i[0]),int(i[1]))
                PointObject_B=Point(int(j[0]),int(j[1]))

                #if they are equal, then True will be returned
                #therefore the if function will execute
                if(PointObject_A.__eq__(PointObject_B) and allow):

                    #remove input "i" from list, self.points
                    self.points.remove(i)

                    #assuming that no points are repeated, the j-loop will now break
                    break


if __name__ == "__main__":
    list_points = []
    n = int(input())
    for x in range(n):
        input_line = input().split(',')

        # --- YOUR CODE HERE ---
        list_points.append(input_line)

    # create the Line instance using the first two points
    myLine = Line(list_points[0], list_points[1])

    # If your methods are defined correctly, the following lines should produce the desired output in the test cases.
    myLine.add_points(list_points[2:-1])
    print(myLine)
    myLine.add_points(list_points)
    print(myLine)
    myLine.remove_points(list_points)
    print(myLine)
```