MINIMUM DISTANCE POINTS

Given 'n' points, design a flowchart or an algorithm and write the Python code to determine the pair of points that are closer. Distance between two points (x1, y1) and (x2, y2) is determined using the formula. Write a function to determine distance between the two points. Consider only two decimal places of distance for comparison.

Input Format

Number of points

x coordinate of point1

y coordinate of point1

x coordinate of point2

y coordinate of point2

...

Output format

Point1 and Point2 that have minimum distance between them

```
def distance(x1, y1, x2, y2):
    import math
    dist = format(math.sqrt(((x1-x2)**2)+((y1-y2)**2)), '.2f')
    return float(dist)

n = int(input())
if n > 0:
    points = []

    for i in range(n):
        x = int(input())
        y = int(input())
        pt = tuple([x, y])
        points.append(pt)

    mini = distance(points[0][0], points[0][1], points[1][0],
points[1][1])
    pos1 = [-1]
    pos2 = [-1]

    for i in range(n):
        for j in range(i+1, n):
            dis = distance(points[i][0], points[i][1],
points[j][0], points[j][1])

            if mini > dis:
```

```
                mini = dis
                pos1[0] = i
                pos2[0] = j
            elif mini == dis:
                pos1.append(i)
                pos2.append(j)

    flag1 = 0
    flag2 = 0

    for i in pos1:
        for j in pos2:
            if distance(points[i][0], points[i][1], points[j][0],
points[j][1]) == mini:
                if flag1 == points[i] and flag2 == points[j]:
                    continue
                print(points[i])
                print(points[j])
                flag1 = points[i]
                flag2 = points[j]
                break

else:
    print('Invalid input')
```

Boundary Conditions

Number of points > 0

Python:

Algorithm:

1. Start
2. Define function to find distance using the distance formula.
3. Read n, and read n co-ordinates of n points.
4. For 0 to n
    a. Find the distance for each pair of points of the n entered.
    b. Find the minimum distance
    c. Note down the positions of the points
5. Print the corresponding point at the position noted down.
6. Stop

Input:

Input n, co-ordinates of n points

Process:

Check the validity of the input.

Find the distance between each of the points and note down the minimum.

Output:

Print the points for which distance is minimum.

COMMON FACTORS

Given a list of four numbers, write a Python code to print the factors of each number in the list, common factors of all the four numbers, factors of the last two numbers which are not factors of the first and the second number in the list. Factors of a number doesn't include 1 but includes itself.

For example, the factors of 12 are {2, 3, 4, 6, 12}.

Input Format

Number1

Number2

Number3

Number4

Output Format

Factors common to all the four numbers in the list as a sorted list

Common factors of last two numbers which are not factors of first and second number in a sorted list

Python:

```
num_list = [0, 0, 0, 0]
factor_list = []

for i in range(4):
    num_list[i] = int(input())
    num_set = set()
    for j in range(2, num_list[i]+1):
        if num_list[i] % j == 0:
            num_set.add(j)
    factor_list.append(num_set)
```

```
inter = sorted(list(factor_list[0] & factor_list[1] &
factor_list[2] & factor_list[3]))
last_inter = sorted(list(((factor_list[2] & factor_list[3]) -
(factor_list[0])) |
                         ((factor_list[2] & factor_list[3]) -
(factor_list[1]))))

print(inter)
print(last_inter)
```

Pseudo Code:

SET num_list AS [0, 0, 0, 0]

SET factor_list AS []

FOR i = 0 TO 4

    READ num

    FIND factors FOR num

    APPEND factors TO factor_list

END FOR

SET inter AS factor_list[0] & factor_list[1] & factor_list[2] & factor_list[3]

SET inter_last AS ((factor_list[2] & factor_list[3]) - (factor_list[0])) | ((factor_list[2] & factor_list[3]) - (factor_list[1]))

PRINT inter, inter_last

END

Input:

Input 4 numbers

Process:

Check the validity of the input.

Find the common factor using set notation and find common factors of 3rd and 4th number not in 1st and 2nd numbers.

Output:

Print inter and inter_last.

STUDENT GRADE

Given details of 'n' students, design a flowchart/algorithm and write the Python code to determine the average marks scored by each student. The details of the student include name, register number, and five marks. Write a function to determine the total marks scored by the student and don't let the function to modify the marks.

Input Format

Number of students

Register number of student1

Five marks of student1

....

Output Format

Details of student as a list with name, register number, marks and total

Python:

```
def total(x1, x2, x3, x4, x5):
    return x1 + x2 + x3 + x4 + x5

stud_list = []

n = int(input())

for i in range(n):
    stud = []
    name = input()
    reg = input()
    m = []
    for j in range(5):
        temp = int(input())
        m.append(temp)
    m = tuple(m)
    tot = total(m[0], m[1], m[2], m[3], m[4])
    stud.append(name)
    stud.append(reg)
    stud.append(m)
    stud.append(tot)
    stud_list.append(stud)

print(stud_list)
```

Algorithm:

1. Start
2. Define function to find total marks
3. Read n
4. In a loop, input student details and call the function to calculate total.
5. Print student list
6. Stop

Input:

Input n, and n students' details.

Process:

Check the validity of the input.

Calculate the total marks.

Output:

Print all the student details.

CHESS BOARD POSITIONS

Given the position of a rook and a queen in a chess board write a Python code to determine the common positions where both rook and queen can be placed in the next move. Each cell in the chess board may be represented by as a tuple (row, col).

For example, if the current position of the rook is (3,1) then the next possible position of the rook are in the same column {(2,1),(1,1),(4,1),(5,1),(6,1),(7,1),(8,1)} or in the same row {(3,2),(3,3),(3,4),(3,5),(3,6),(3,7),(3,8)}. If the queen is in the position (5,3) then it can be placed in the same row {(5,1),(5,2),(5,4),(5,5),(5,6),(5,7),(5,8)} or same column {(1,3),(2,3),(3,3),(4,3),(6,3),(7,3),(8,3)} or along the diagonal of the current position {(6,4),(7,5),(8,6),(4,2),(5,1),(6,2),(7,1),(4,4),(3,5),(2,6),(1,7)}. Then the common cells for next move are {(3,5), (3,3), (5,1), (7,1)}.

The output is a set of common board positions where both queen and rook can be placed. The positions must be printed in sorted order, sort it by row when rows are same sort it by column.

(*Hint:* Use built-in function to sort the values)

Input Format

Row position of rook

Column position of rook

Row position of queen

Column position of queen

Output Format

Common position1

Common position2

...

Python:

```
rkr = int(input())
rkc = int(input())
rk = tuple([rkr, rkc])

qr = int(input())
qc = int(input())
q = tuple([qr, qc])

# All possible positions of rook.

rook = set()

# By column
x = rkc
while x < 8:
    x += 1
    rook.add(tuple([rkr, x]))

x = rkc
while x > 1:
    x -= 1
    rook.add(tuple([rkr, x]))

# By row
x = rkr
while x < 8:
    x += 1
    rook.add(tuple([x, rkc]))

x = rkr
while x > 1:
    x -= 1
    rook.add(tuple([x, rkc]))

# All possible positions of queen.
```

```
queen = set()

# By column
x = qc
while x < 8:
    x += 1
    queen.add(tuple([qr, x]))

x = qc
while x > 1:
    x -= 1
    queen.add(tuple([qr, x]))

# By row
x = qr
while x < 8:
    x += 1
    queen.add(tuple([x, qc]))

x = qr
while x > 1:
    x -= 1
    queen.add(tuple([x, qc]))

# By diagonal
x = qr
y = qc
while x < 8 and y < 8:
    x += 1
    y += 1
    queen.add(tuple([x, y]))

x = qr
y = qc
while x < 8 and y > 1:
    x += 1
    y -= 1
    queen.add(tuple([x, y]))

x = qr
y = qc
while x > 1 and y > 1:
    x -= 1
    y -= 1
    queen.add(tuple([x, y]))
```

```
x = qr
y = qc
while x > 1 and y < 8:
    x -= 1
    y += 1
    queen.add(tuple([x, y]))

common_raw = list(queen & rook)
common = sorted(common_raw)

for i in common:
    print(i)
```

Pseudo Code:

READ rkr, rkc, qr, qc

FIND all possible positions of rook

FIND all possible positions of queen

FIND intersection of the two

PRINT intersection

END

Input:

Input rook and queen positions.

Process:

Check the validity of the input.

Find all possible positions of queen and rook. Find their intersection.

Output:

Print the intersection of positions.