Assessment 4: Triangle Classification

CSCI 128

Problem

Given three points (x, y) in a coordinate plane, determine if they form a triangle. If they form a triangle, then classify the triangle by both its side lengths and angles.

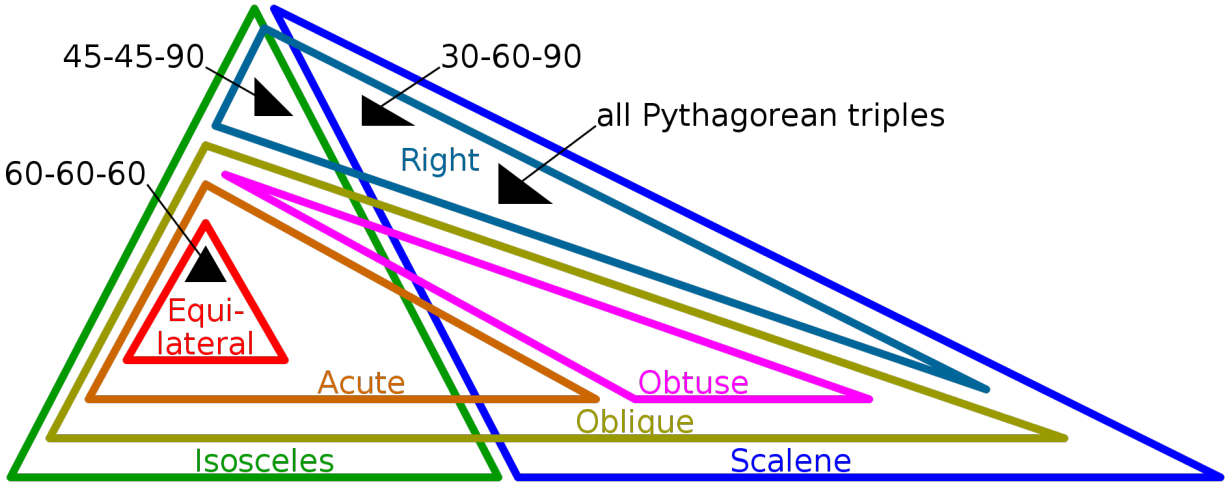

Figure 1: Euler diagram of triangle types.
Source: Wikipedia (cc license).

See the Output section for the exact classification details.

Input

There are three total lines of input. Each line of input contains two space-separated floats denoting the x and y coordinates of a point on the triangle.

Output

You will print 2 lines of output. Line 1 will be the lengths of the sides sorted in ascending order.

Your list of sides should also be rounded to 4 decimal places **using the `round()` function** (check the hints for more info about this!).

For example:

```
OUTPUT [11.0, 12.0, 14.0]
```

```
OUTPUT [11.4562, 11.546, 12.0]
```

Your next line will be the classification of the points, subject to the following criteria.

You should use the side lengths list rounded to 4 decimal places **with the `round()` function** to determine the classification.

- If there are any duplicate points, output "Duplicate Point". For instance:

  ```
  OUTPUT Duplicate Point
  ```

- If the three points are collinear, then output "Collinear". For instance:

  ```
  OUTPUT Collinear
  ```

- Otherwise, the three points form a triangle. Classify the triangle by its internal angles (Acute, Right, Obtuse). Then classify the triangle by its side lengths (Equilateral, Isosceles, or Scalene).

  For instance, if we had a scalene triangle with an obtuse angle, we would output:

  ```
  OUTPUT Obtuse Scalene Triangle
  ```

  As another example, if we had an equilateral triangle (recall that all angles in an equilateral triangle are acute) we would output:

  ```
  OUTPUT Acute Equilateral Triangle
  ```

Hints

1. You can sort a list in-place using the `sort()` function. For example:

   ```
   lst1 = [8, 9, 7]
   lst1.sort()
   print(lst1) # [7, 8, 9]
   ```

   By default, `sort()` will order the elements of a list in ascending order.

2. Distances can also be used to test for collinearity. If three points occur on the same line, then the sum of the two shorter distances will exactly equal the largest distance.

3. Consider the following for determining internal angles of a triangle:
   Let S be the shortest side, with M and L the middle and longest sides respectively.
   If $S^2 + M^2 < L^2$ then the triangle is **Obtuse**
   If $S^2 + M^2 = L^2$ then the triangle is **Right**
   If $S^2 + M^2 > L^2$ then the triangle is **Acute**

4. You may want to to take a square root as part of this assessment. Recall from your math classes that $\sqrt{x} \equiv x^{\frac{1}{2}} \equiv x^{0.5}$.

5. `round()` takes in 2 arguments, the float to round and the number of decimal places to round to.

   For example:

```
pi = 3.14159265359
rounded_pi = round(pi, 4)
print(rounded_pi) # 3.1416
```

Reflection

This problem can be most easily related to mathematics. You may have even used, or wished you had, a program like this in geometry class. While the content used here should be familiar, it was likely a new and challenging problem for you to change your mental model of how triangles are organized into unambiguous computer instructions.

For reflection this week, describe how you wrote your solution program. Reflecting on how you wrote your code is an important step in improving that process. You don't have to answer all of the following questions, but they may help organize your thoughts.

- Did you do any writing before starting to code? If yes, what did it look like? Maybe you did some math to remember your geometry fundamentals, or maybe you tried writing some pseudocode?
- Where in your code did you decide to check for duplicate points and co-linearity? Why did you pick that position?
- What kind of triangle did you decide to check for first? Did you have a reason to do that?
- Did you go back and re-write your code at any point? If yes, what made you decide to do that?
- How did you decide that your program was finished?

Sample Executions

```
P1> 0 0
P2> 0 4
P3> 3 0
OUTPUT [3.0, 4.0, 5.0]
OUTPUT Right Scalene Triangle
```

```
P1> 0 0
P2> 12 3
P3> 23 66
OUTPUT [12.3693, 63.9531, 69.8928]
OUTPUT Obtuse Scalene Triangle
```

```
P1> 100 0
P2> 0 0
P3> 50 86.6025403784
OUTPUT [100.0, 100.0, 100.0]
OUTPUT Acute Equilateral Triangle
```

```
P1> 0 2
P2> 0 2
P3> 3 1
OUTPUT [0.0, 3.1623, 3.1623]
OUTPUT Duplicate Point
```

```
P1> 1 1
P2> 0 0
P3> 2 2
OUTPUT [1.4142, 1.4142, 2.8284]
OUTPUT Collinear
```