Daily Coding Problem #236

Problem

This problem was asked by Nvidia.

You are given a list of `N` points (x_1, y_1), (x_2, y_2), ..., (x_N, y_N) representing a polygon. You can assume these points are given in order; that is, you can construct the polygon by connecting point `1` to point `2`, point `2` to point `3`, and so on, finally looping around to connect point `N` to point `1`.

Determine if a new point `p` lies inside this polygon. (If `p` is on the boundary of the polygon, you should return `False`).

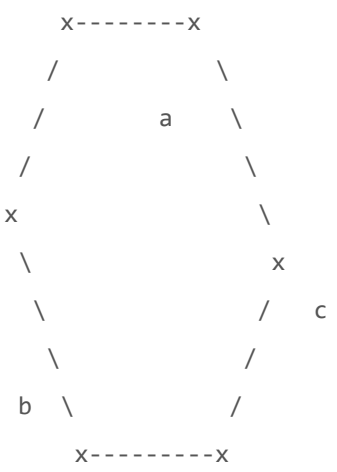
Solution

At first it may appear that we need to carry out a lot of complicated math to answer this problem.

In fact, there is a relatively straightforward solution, called the ray-casting algorithm. The idea is to draw a horizontal line starting from the point and

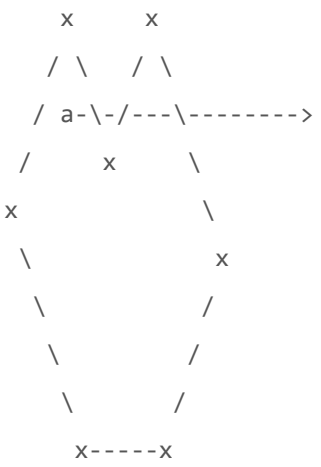
extending rightward. If this line crosses an odd number of line segments of our polygon, it is on the inside; if not, it is on the outside.

Let's look at an example to see how this works.



Here we have a six-sided polygon with nearby points a, b, and c. The point a is inside the polygon, so it only takes one intersection to get out. Point b begins to the left of the polygon, so it will need to cross once to get inside, and again to get out. Finally, point c begins to the right of the polygon, so there will be no intersections at all.

Even for non-convex polygons, this will hold true. For example, in the following figure, the horizontal line extending from point a crosses a non-convex region, hitting the polygon at two points, and then crosses once more, resulting in an odd number of intersections.



To implement this, we can check, for each side, whether the ray cast from the point intersects the side. If there are an odd number of sides with this property, we know the point is inside the polygon.

We can find the intersection point with some algebra. Recall that we can represent a line as `y = mx + b`, where `m` is the slope and `b` is the intercept. For each side, then, we can calculate the slope and intercept from the given vertices, and plug in the `y` coordinate of our point, to solve for the intersection (x_p, y_p) To ensure that this intersection point is valid, we must check to see that:

- x_p is to the right of our point, and
- y_p lies between the `y`-coordinates of each side's vertices.

We will also make use of a `Point` class, to help clarify some of the algebra.

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

def create_sides(points):
    sides = [(points[-1], points[0])] + list(zip(points, points[1:]))
    return [(Point(*a), Point(*b)) for (a, b) in sides]

def intersects(p, side):
    p1, p2 = side

    # Get the slope and intercept of the side. Check for zero and undefined
slopes.
    dy, dx = (p2.y - p1.y), (p2.x - p1.x)
    if dx == 0.0:
        return 1 if p.x < p1.x and min(p1.y, p2.y) <= p.y <= max(p1.y, p2.y)
else 0
    if dy == 0.0:
        return 0

    slope = dy / dx
```

```
    intercept = p1.y - slope * p1.x

    # Plug in the y-coordinate of our point and solve for the intersection.
    intersection = Point((p.y - intercept) / slope, p.y)

    # Check to see if the intersection is valid before returning.
    if p.x <= intersection.x and min(p1.y, p2.y) <= intersection.y <= max(p1.y,
p2.y):
        return 1
    else:
        return 0

def check_inside(p, polygon):
    p = Point(*p)

    count = 0
    sides = create_sides(polygon)
    for side in sides:
        count += intersects(p, side)

    if count % 2 == 1:
        return True
    else:
        return False
```

Since we perform a constant number of operations per side, this algorithm is `O(N)`, where `N` is the number of sides.

Press