ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικό και Καποδιστριακό
Πανεπιστήμιο Αθηνών

Name: Foteini Tsavo

Registration Number: 1115201500206

Emails: sdi1500206@di.uoa.gr / fotinicavo97@gmail.com

Exercise 1: Implement an algorithm that takes as input three points in the plane, checks that they form a triangle and whether the interior of the triangle contains the origin (0, 0) or not.

Firstly, we have to find the area of triangle that is formed by the three points given as input. If the result is not zero, it means that the three points are not collinear, therefore they are able to form a triangle. The area of the triangle is given by the formula:

$$\left| \frac{1}{2} \det(\overrightarrow{AB}, \overrightarrow{AC}) \right|$$

Let $A = (x_1, y_1),\ B = (x_2, y_2),\ C = (x_3, y_3)$ then the determinant will be the following:

$$\frac{1}{2} \begin{pmatrix} x_1 - x_2 & x_2 - x_3 \\ y_1 - y_2 & y_2 - y_3 \end{pmatrix}$$

Now, we apply the determinant formula:

$$|(x_1 - x_2)(y_2 - y_3) - (y_1 - y_2)(x_2 - x_3)| \neq 0$$

Another way of proving this, would have been to find the equation of two vertices and then check if the third point is present on the line. If it is, it means that the points are collinear, so they lie on the same straight line. Otherwise, they can form a triangle. We did not choose this way, because it is simpler to implement the first method and as a result we will have a faster algorithm and cleaner code.

Furthermore, if we desire to find whether the origin (0,0) lies inside the triangle or not, we have to do some certain computations:

1. We calculate the area of the triangle by using the coordinate geometry formula that we also, used above.
2. To check if the origin lies inside this triangle, we have to create three new triangles that contain the origin combined with two of the points each time. For instance if our first triangle is supposed to be ABC, then the three new ones will be OAB,OAC and OBC We will use the same formula to compute the area of the triangle and we will have result1, result2, result3 respectively.

3. If the point (0,0) lies inside then the sum of the results we have calculated will be equal to 0. Else the point does not lie inside.

Explaining the code:

A class named Points holds the x value and y value that the user inputs from the console. Every time we would like to construct a point, we will call the function Fill, which is responsible for creating points and adding them to a list of Points objects named triangle. The function IfTriangle will take this list as a parameter and do the proper calculations based on the coordinate geometry formula. The printAnswer parameter is 0 when we do not desire to print anything but just want to extract the value. We set it to 1 only one time,in order to print the answer whether the points are part of a triangle or of a straight line. If the conclusion is that the vertices can definitely form a triangle, then we continue on finding if the origin lies inside this one. The function OriginIncluded is responsible for answering if the origin is inside or not. We would like to specify that this program will work for any point, not only for the origin. To continue with, we will make the right substitutions in the right positions of our lists. Every time we copy the last list to the new one and replace the element in the right position so as to have the right results. After that we do the sum and if it equal to 0 it means that (0,0) is part of the triangle's area.

How to run the program:

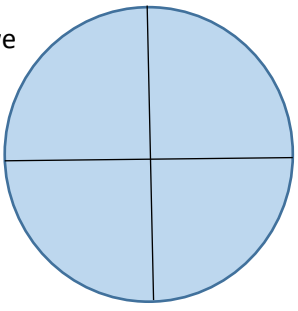
$python exercise1.py

After running the program, it will appear a line that requests from user to insert two floats x and y. This will appear three times because three points are required. After the insertion, the program will do the calculations and return messages: "Yes" or "No" if the points can create a triangle and "Origin is inside!" or "Origin is not inside!" depending the position of the origin (0,0).

Exercise 2: Given a circle of radius r in the plane with (0, 0) as center, implement an algorithm that finds the total lattice points on the circumference. Lattice Points are points with integer coordinates.

All the points of the circle with origin (0, 0), should satisfy the following equation: $x^2 + y^2 = r^2$. In our case, we are searching for integer points, only, on the circumference of it.

As we can see, those two straight lines may hold the points we are searching for. Consequently , if we find one point (x, y), then its combinations will also exist. Therefore, 3 more results are added to



our list (-x, y), (x, -y), (-x, -y). The only exception that may occur is when we have (r,0), (0,r) and that is because there is no negative 0.

```
Initialize our combination ← 4
Set x ← 1
While x ≠ r:
        If r*r – x*x is perfect square:
                combination = combination + 4
        End if
        x = x + 1
End while
```

Explaining the code:

As well as our previews example, we have used the class Points here, to represent the origin of the circle, and also, added another class Circle that consists of the origin and the radius. The radius is specified from the user. The function laticePoints is the one that answers the question of how many lattice points can be found in this particular circle. The initialization of the combinations with 4 is made because in our case, there are 4 points that we already know of: (0,r), (r,0), (-r,0), (0,-r). After this step we continue with a loop and start from x = 1 , until the value of x will reach r-1. Every time, we check if the r*r – x*x will result in a y that is integer. In order to distinguish if y value is an integer or a float, a function is_integer() has been used. It is a function of python that is able to understand whether the current float is integer or is decimal. This way seems faster than computing if it really is a perfect square or not.

How to run the program:

$python exercise1.py

The program will request the length of the radius from the user. After hitting enter, the answer will appear on the console.

Exercise 4: Implement the gift wrap algorithm for computing the convex hull of a finite set of points in the plane .

Generally, the gift wrap algorithm or else Jarvis's Algorithm, is about finding the convex hull of the set of points in the plane, which is the smallest convex polygon that contains all the points. We start by adding to the set of the hull the leftmost point, that is the point with the smallest x value. In case there are two points with equal minimum x values, then we prefer the point whose y value is greater than the other. After this step, we keep wrapping

points in counterclockwise direction until we do not come back to the initial point. Every next point is chosen with the criteria such as beating others in counterclockwise orientation, that means we choose the leftmost point.

Counterclockwise turns:

$$\begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = a_x b_y - a_y b_x + a_y c_x - a_x c_y + b_x c_y - c_x b_y$$

$$= (b_x - a_x)(c_y - a_y) - (c_x - a_x)(b_y - a_y)$$

If the value above equals 0, this means that the three points are collinear. Another case may be a value greater than 0, which means that the direction is clockwise, or else a right turn. Otherwise, the value will be negative and this is supposed to represent a counterclockwise direction, that is the one we are interested in (left turn).

```
L ← leftmost point of the set s
p ← l
hull ← {}
Do:
        Add p to hull
        q = (p+1)% n      //n is the number of points in set s
        For i = 0 to n:
                point q is the point such that the triplet (p, q, r) is counterclockwise for any r
        end For
        p = q             //Set p as q for next iteration
While p != L              //we do not come to first point
```

Explaining the code:

Another class has been added this time. Dataset contains a list of the points that were added randomly. This particular class has a function that finds the leftmost element of the dataset, and by that we mean the point that has the smallest x. If there are more than one point with this minimum x, we are obligated to check their y coordinated and return the one that has the greatest y value. Furthermore, the convexhull function is the one that the magic happens. We work with the positions of the points, but finally we return a list of points, hull list. In order to find which is the most appropriate point to be added to hull next, the function Counterclockwise is called and it returns with value 1 the most left element.

How to run the program:

$python exercise4.py

The program will run immediately, as there is no user input but someone can change the input from the inside :

```
for i in range(0,5):
    allPoints.append(Points(randint(0,10),randint(0,10)))
```

Exercise 3: Implement the incremental 2D algorithm for computing the convex hull of a finite set of points in the plane.

The Incremental 2D Convex Hull Algorithm has the complexity of O(nlogn). This algorithm divides the problem into computing the top and bottom parts of the hull separately. We iterate from the leftmost vertex to the rightmost vertex, add them to the hull, and check backwards for concavities. Specifically, we use sorted lists based on x values, and if x-s are equal than we have to sort for y values too. Generally, we can think of this algorithm as if walking the boundary of a polygon on a clockwise direction, on each vertex there is a turn left, or right. On the convex hull polygon, this turn will always be a right turn.

The basic step of the incremental algorithm is the update of the top of the hull after inserting a point p_i. The next thing we do is to check whether the triplet that is made from the last three points of the upper list make a right turn. If this is true our work is done. On the

---

sort(s) by x-coordinate and by y-coordinate //with s we refer to the dataset
upper ← $\{p_1, p_2\}$
For i=2 to n:
    add p_i to upper list
    While |upper| > 2 and the three last points of upper do not make right turn:
        Delete the point next to the last one
    End while
End for
//Do the same for the downlist, downlist has the reverse sorting
Add downlist to upper

---

other hand , we should remove the element next to the last of the list. We continue this until the last three points make a right turn, or if the list is left with only two members.

Explaining the code:

Firstly, in our class Dataset, we have two functions that sort the points of the list with a lambda expression. Moreover, the listConvexHull function responsible for computing the upper and the lower hulls. It just needs the list of points sorted in the appropriate way. After that, the IncConvexHull function adds the lower to the upper part and returns the final convex hull.

How to run the program:

$python exercise3.py

The program will run immediately, as there is no user input but someone can change the input from the inside.