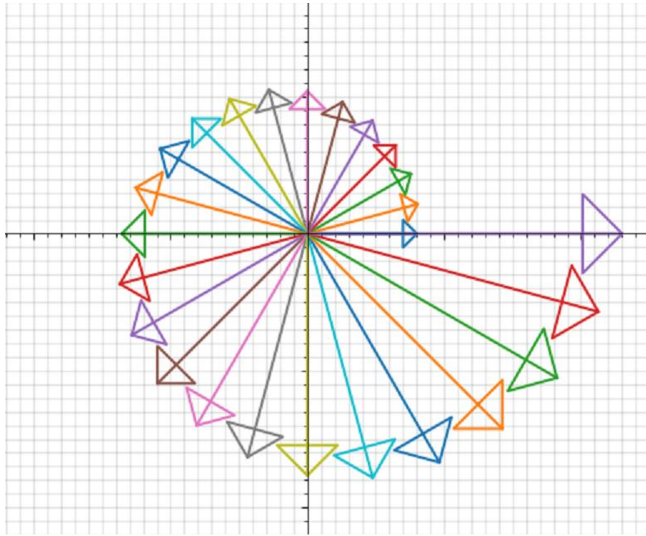


POLAR COORDINATES WITH PYTHON: VOLUME TWO

Plotting XY and Polar Coordinates

ABSTRACT

XY and Polar coordinates. Point plotting code examples using Python

Armand Laurino

[Course title]

Contents

Introduction 2

Trig Tangent and the Inverse 2

Common Classes and Modules 4

XY Coordinates 18

  We now play! 18

    Polygon Shift 22

    Example: multiple shifts 24

    Example: triangle shift 26

    More Shift methods 27

    Multiple triangle shifts 29

    Multiple triangle shifts off center 31

Polar Coordinates 33

  We now play 37

    Wine bottle rotated 37

    Rotations on one XY Plane 38

    Shift and rotate example 41

    Rotate and shift example 43

    Spin an arrow 45

    Spin and enlarge an arrow 46

    Stairway to nowhere 48

Almost a Circle 50

  Regular Polygon Builder 50

  Math Sequences 56

  Almost a circle 58

Appendix: Common Classes and Scripts 61

  Class Point 61

  Class Points 62

  Class XY Plane 63

  Class AngleMeasure 64

  Class PolarPoint 65

  Module XY Utils 66

  Class XYDegreeSpinner 68

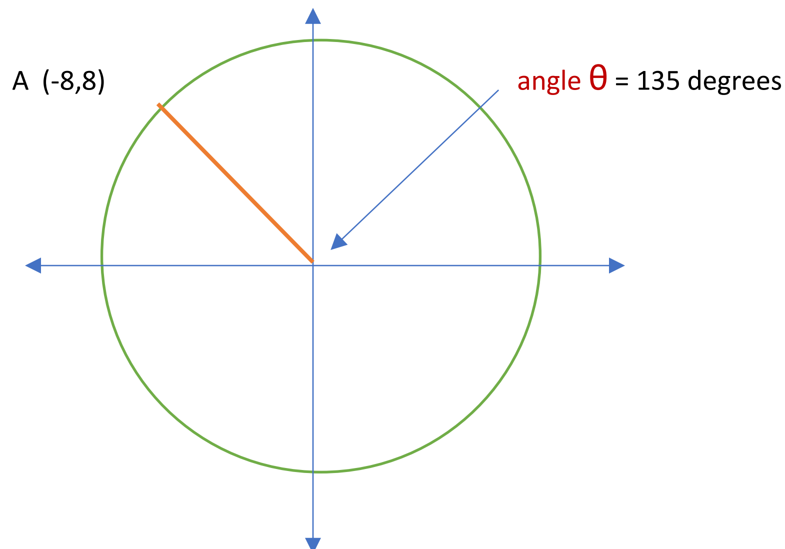
Introduction

After reviewing some basic math concepts in Volume One we now do some ‘primitive’ coding. The functionality that we will build is already available in numerous free Python libs. Our intent here is NOT to replace such libs but to ignore them and ‘roll our own.’ The ‘roll you own’ rational is to write some ‘throw-away’ code as a learning experience - this is strictly for beginners. This article is NOT intended for advanced and/or experienced python programmers.

I emphasize for ‘professional coding work’ do some research and use one of the dependable and well-reviewed libs!!!

Trig Tangent and the Inverse

We look at angles 45 and 135 degrees in a circle of radius R For any angle θ and its point of intersection (X, Y) with our circle we have tangent(θ) = Y/X



Here is a mini-Python snippet that calculates some tangent function ratios for us!

```
degreesTuple = (45, 135, 225, 315, -45)
for degrees in degreesTuple:
    # convert angle from degrees to radians
    rads = math.radians(degrees)
    # get tangent of angle
    tan = math.tan(rads)
    # round to 3 decimal places
    tan = round(tan,3)
    print("Tan of " + str(degrees) + " -> " + str(tan))
```

Output:

Tan of 45 -> 1.0

Tan of 135 -> -1.0

Tan of 225 -> 1.0

Tan of 315 -> -1.0

Tan of -45 -> -1.0

If we use our conventional X and Y variables we see that the equation: Y = tan(X) is a ***function*** that is **NOT *one to one.*** If we look closer at the python math library we notice an inverse tan function. Remember! A ***function*** cannot have an **inverse *function*** unless it is ***one to one***! Hum! Looks like python could care less!

More python!

```
tanValues = (-1.0, 1.0)
for tanValue in tanValues:
    rads = math.atan(tanValue)
    # convert angle from rads to degrees
    degrees = math.degrees(rads)
    # round to 3 decimal places
    degrees = round(degrees,3)
    print("Arc Tan of " + str(tanValue) + " -> " + str(degrees))
```

Output:

Arc Tan of -1.0 -> -45.0

Arc Tan of 1.0 -> 45

We will see and address this issue a bit later. In the mean-time we look at some code.

Common Classes and Modules

File name	***Class or module***	***Function***
Point.py	Class Point	An x,y coordinate pair (x,y)
Points.py	Class Points	An ordered collection of Points
XYPlane.py	Class XYPlane	The XY Plane
AngleMesaure.py	Class AngleMesaure	To distinguish degrees vs. radians
PolarPoint.py	Class PolarPoint	A polar coordinate (distance, angle)
XYUtils.py	Module	Various XY and Polar utilities
Spinner.py	Class XYDegreeSpinner	Spins (rotates) a Points collection

Full code listings are in our appendix -

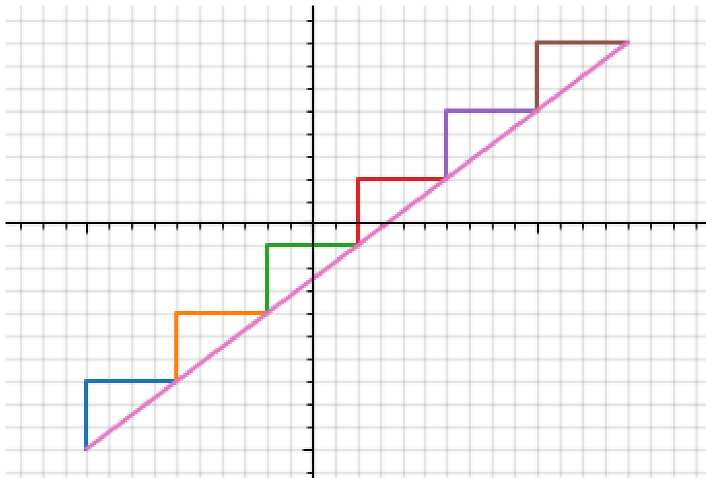
Stairway to nowhere

We build stairs

Code

```
import matplotlib.pyplot as plt
import XYUtils
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()
    points = Points()
    firstPoint = Point(-10, -10)
    points.append(firstPoint)
    points.append(Point(-10, -7))
    points.append(Point(-6, -7))

    xyPlane.plot(points)

    J = 0
    deltaX = 4
    deltaY = 3

    deltaXSum = deltaX
    deltaYSum = deltaY

    while J < 5:
        points = XYUtils.xyShift(points, deltaX, deltaY)
        deltaXSum = deltaXSum + deltaX
        deltaYSum = deltaYSum + deltaY
        xyPlane.plot(points)
        J = J + 1

    lastPoint = points.asList()[-1]
    firstLast = Points()
    firstLast.append(firstPoint)
    firstLast.append(lastPoint)
    xyPlane.plot(firstLast)

    d = XYUtils.distance(lastPoint, firstPoint)
    print("First point: " + firstPoint.str() + " Last point: " + lastPoint.str())
    print("Delta X sum: " + str(deltaXSum) + " Delta Y sum: " + str(deltaYSum) + " Hypotenuse: 
" + str(d))

    plt.show()

# eof
```

Output



First point: (-10,-10) Last point: (14,8)
Delta X sum: 24 Delta Y sum: 18 Hypotenuse: 30.0

Notes:

Each of our steps is a 3-4-5 Pythagorean triplet because our x delta is 4 and y delta is 3!

The distance (from our first point to our last point is 30

18, 24, 30 form a right triangle similar to our 3,4,5, right triangles. Pythagorean theorem

182 + 242 = 302
324 + 576 = 900

We have 6 steps. Remember our ratio discussion re similar triangles in Volume One!

18 / 6 = 3   24/6 =4   30/6 = 5

A Circle

Regular Polygon Builder

We begin by plotting a regular polygon centered at the origin.

Definition: A polygon is regular when all angles are equal, ***and*** all sides are equal

We jump right into the code.

We code a regular polygon builder. The builder requires two inputs

- The regular polygon's circumradius
- The number of sides N where N >= 3

To keep our examples small enough to plot we use 8 for our circumradius.

Code with default values. Needed to build a regular polygon

```
class RegularPolygonBuilderPacket:
    def __new__(cls, *args, **kwargs):
        return super().__new__(cls)

    def __init__(self):
        self.circumRadius = 8
        self.numSides = 3
# END
```

Regular polygon builder

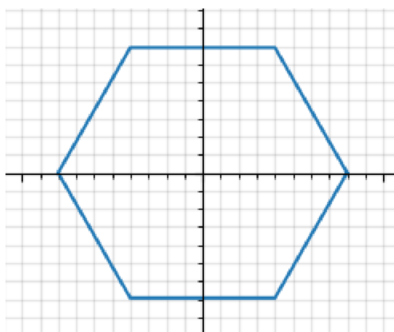
```
import math
import XYUtils
from AngleMeasure import AngleMeasure
from PolarPoint import PolarPoint
from Points import Points
from RegularPolygonBuilderPacket import RegularPolygonBuilderPacket


class RegularPolygonBuilder:
    def __new__(cls, *args, **kwargs):
        return super().__new__(cls)

    def __init__(self, buildPacket: RegularPolygonBuilderPacket):
        self.circumRadius = buildPacket.circumRadius
        self.numSides = buildPacket.numSides
        self.side = 0
        self.apothem = 0
        self.area = 0
        self.perimeter = 0
        self.angle = 0
        self.xyPoints = Points()
        self.angle = 360.0 / self.numSides

        angleMeasure = AngleMeasure()
        angleMeasure.useDegrees()

        ppAngle = 0
        while ppAngle <= 360:
            pPoint = PolarPoint()
            pPoint.distance = self.circumRadius
            pPoint.angle = ppAngle
            xyPoint = XYUtils.polarPointToXY(pPoint, angleMeasure)
            self.xyPoints.append(xyPoint)
            ppAngle = ppAngle + self.angle

        xyList = self.xyPoints.asList()
        p1 = xyList[0]
        p2 = xyList[1]
        self.side = XYUtils.distance(p1, p2)

        self.perimeter = self.numSides * self.side
        self.apothem = math.sqrt(self.circumRadius*self.circumRadius-self.side/2*self.side/2)
        self.area = self.perimeter * self.apothem / 2.0

# END
```

We now exercise the builder. We build a **Hexagon - a six-sided** regular polygon. Change the 6 to other integer values to play!

```
import matplotlib.pyplot as plt
from RegularPolygonBuilder import RegularPolygonBuilder
from RegularPolygonBuilderPacket import RegularPolygonBuilderPacket
from XYPlane import XYPlane

if __name__ == '__main__':
    xyPlane = XYPlane()

    builderPacket = RegularPolygonBuilderPacket()
    builderPacket.numSides = 6
    builderPacket.circumRadius = 8

    builder = RegularPolygonBuilder(builderPacket)

    print("Polygon sides : " + str(builder.numSides) + " Side length: " + str(builder.side))
    print("Polygon perimeter : " + str(builder.perimeter))
    print("Polygon area : " + str(builder.area))
    print("Polygon apothem : " + str(builder.apothem))
    print("Polygon angle : " + str(builder.angle))

    xyPlane.plot(builder.xyPoints)
    plt.show()

# END
```

Output

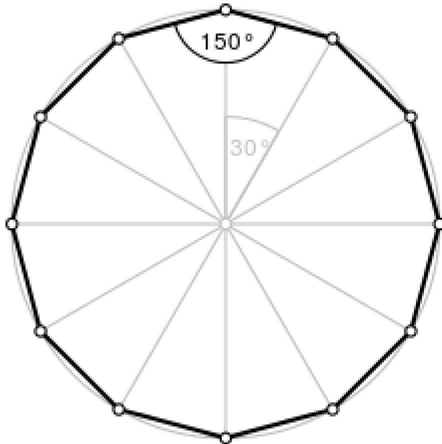
Polygon sides : 6 Side length: 7.999997202499511
Polygon perimeter : 47.999983214997066
Polygon area : 166.27683876329706
Polygon apothem : 6.928204037844151
Polygon angle : 60.0



Here is the polygon stats with 12 sides

Polygon sides : 12 Side length: 4.141105557698331
Polygon perimeter : 49.693266692379964
Polygon area : 192.00003598027567
Polygon apothem : 7.727406498302002
Polygon angle : 30.0

I stole this diagram from the web!



We now check the python math in our builder. We inspect one of the above isosceles triangles (they are all congruent)

Triangle

Apothem is triangles altitude

30 degrees

15 degrees

polygon sides
length = 8

apo

length 8

90 degrees

75 degrees

75 degrees

The apothem forms two right triangles from our isosceles triangle. We display one of the two. Consider the **15-degree angle above** (you can also use the 75-degree angle!) Because the triangle is a right triangle we can use our calculator to calculate the following.

sine(15) = 0.25881904510252076234889883762405 = opposite/ hypotenuse
0.25881904510252076234889883762405 = opposite/8
2.0705523608201660987911907009924 = opposite

cosine(15) = 0.9659258262890682867497431997289 = adjacent/hypotenuse
0.9659258262890682867497431997289 = adjacent/8
7.7274066103125462939979455978312 = adjacent

Looks closely, the triangle opposite side is ½ the polygon side
Polygon side = 2.0705523608201660987911907009924 * 2
Polygon side = 4.1411047216403321975823814019848

The triangle adjacent side is the polygon apothem
Polygon apothem = 7.7274066103125462939979455978312

Our polygon has 12 sides

Polygon perimeter = 12 * 4.1411047216403321975823814019848
Polygon perimeter = 49.693256659683986370988576823817

In volume One we determined the formula for area of a regular polygon
area = perimeter * apothem /2

Polygon Area = 49.693256659683986370988576823817 * 7.7274066103125462939979455978312/ 2
Polygon area = 192

	My calculator	**My Python output**
side	4.1411047216403321975823814019848	4.141105557698331
perimeter	49.693256659683986370988576823817	49.693266692379964
area	192	192.00003598027567
apo	7.7274066103125462939979455978312	7.727406498302002

Math Sequences

We take a bit of a detour here. And talk about sequences. A sequence of numbers is an ordered list of numbers. The sequence usually has a pattern. Because the sequence is ordered we can list them and count them!

Example sequence A

Sequence : { 5, 10, 15, 20, 25…}

I ended our list with a … since we all can see the simple pattern - at least I hope we can!

Notice that as we move thru this sequence from left to right the element numbers get bigger and bigger!

Example sequence B

Sequence (2.9, 2.99, 2.999, 2.9999…}

Notice that as we move thru this sequence from left to right the element numbers get bigger and bigger! For example, it is correct to say that as we move from left to right → the elements get closer and closer to 4

2.9 3 4

The elements of our sequence move along the RED segment of our number line in a left to right direction so that they get closer and closer to 4. They also get closer and closer **to 5 and 3 and 6 and 10.9** and an infinite number of other numbers! They do NOT move closer to 1. In fact, they move farther and farther away from 1!

We looked **at 4 and 3 and 5 and 6 and 10.9** above. Now consider the infinite set of numbers consisting of ALL the numbers that our sequence is moving closer to!

Question what is the smallest number in our set? Excuse me for asking an 'easy question'! Of course, our answer is 3!

Pick any number X 'really close' to 3 on its left side (X < 3) X = 2.99999999992999919191919 will do. As our sequence moves along from left to right it will eventually 'pass' X; in fact, there are an infinite number of sequence elements in here! In the world of Mathematics, the word 'close' is extremely relative! The number line is VERY dense - infinitely so!

X 3

Our sequence B is said to ***converge to*** 3. Specifically, it ***converges towards 3 from the left***; every sequence element is < 3. The number 3 is NOT an element of our sequence

We can use the notation SN for the Nth element in our sequence, e.g., S4 = 2.9999
Now we use the following conventional notation for the convergence. We say that **sequence B *approaches 3 from the left***

$$\text{Limit}_{N \to +\infty} (SN) = 3$$

We say that **sequence A *approaches positive infinity***

$$\text{Limit}_{N \to +\infty} (SN) = +\infty$$

Another sequence example

SN = {1/101, 1/102, 1/103, 1/104, …} here SN = 1/10N

This sequence ***approaches 0 from the right***

$$\text{Limit}_{N \to +\infty} (SN) = 0$$

A circle

We take the above Python code and remove the point plotting

```
import matplotlib.pyplot as plt
from RegularPolygonBuilder import RegularPolygonBuilder
from RegularPolygonBuilderPacket import RegularPolygonBuilderPacket
from XYPlane import XYPlane

if __name__ == '__main__':
    xyPlane = XYPlane()

    builderPacket = RegularPolygonBuilderPacket()
    builderPacket.numSides = 64
    builderPacket.circumRadius = 8

    builder = RegularPolygonBuilder(builderPacket)

    print("Polygon sides : " + str(builder.numSides) + " Side length: " + str(builder.side))
    print("Polygon perimeter : " + str(builder.perimeter))
    print("Polygon area : " + str(builder.area))
    print("Polygon apothem : " + str(builder.apothem))
    print("Polygon angle : " + str(builder.angle))

   # xyPlane.plot(builder.xyPoints)
   #  plt.show()

# END
```

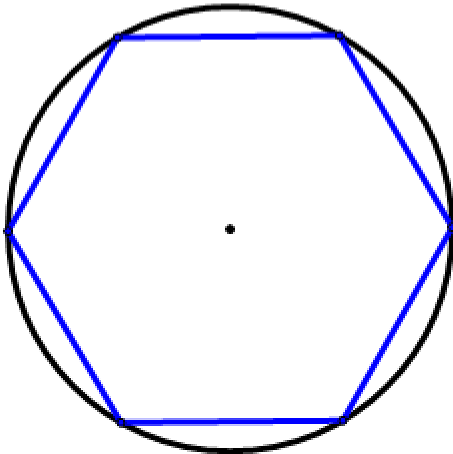
I changed my rounding in XYUtils.py

```
ROUND_DECIMALS = 16
```

We perform multiple runs and get the following outputs

Sides N	Angle (degrees)	Apo	Perimeter	Area
64	5.625	7.990363649641379	50.24529851127605	200.73910339494012
128	2.8125	7.997590549569634	50.260436014924366	200.9811940451042
256	1.40625	7.999397614713156	50.26422081830882	201.04174405969746
512	0.703125	7.999849402260809	50.26516704587346	201.05688327323526
1024	0.3515625	7.9999623504766095	50.26540360443455	201.06066818349385
2048	0.17578125	7.999990587613615	50.265462744179175	201.06161441773813
4096	0.087890625	7.999997646903058	50.26547752912178	201.06185097671639
8192	0.0439453125	7.999999411725743	50.26548122535767	201.06191011648636

If we inscribe any one of the above regular polygons we can visually compare them to the surrounding circle of Radius 8. Our polygons have a bunch more sides than our diagram below BUT you should get the picture!



The surrounding circle has a radius of 8

```
builderPacket.circumRadius = 8
```

Circle details:

circle circumference = Pi * diameter
circle circumference = Pi * 16

circle circumference ≈ 3.14159265358979323846264338327950 * 16

circle circumference ≈ 50.2654824574366918

circle area = Pi * radius * radius

circle area ≈ 3.14159265358979323846264338327950 * 64

circle area ≈ 201.06192982974676726161

Here is our polygon table from above

Sides N	Angle (degrees)	Apo	Perimeter	Area
64	5.625	7.990363649641379	50.24529851127605	200.73910339494012
128	2.8125	7.997590549569634	50.260436014924366	200.9811940451042
256	1.40625	7.999397614713156	50.26422081830882	201.04174405969746
512	0.703125	7.999849402260809	50.26516704587346	201.05688327323526
1024	0.3515625	7.9999623504766095	50.26540360443455	201.06066818349385
2048	0.17578125	7.999990587613615	50.265462744179175	201.06161441773813
4096	0.087890625	7.999997646903058	50.26547752912178	201.06185097671639
8192	0.0439453125	7.999999411725743	50.26548122535767	201.06191011648636
Our surrounding circle				
		Radius	Circumference	Area
		8	50.26548245743669	201.0619298297467672

We can treat each green column in our table as a sequence of numbers.

The apothems are getting larger and larger and are all < 8 (the circle radius)

The perimeters are getting larger and larger and are all < circle circumference

The areas are getting larger and larger and are all < circle area

Using the 'limit' notation we see:

Limit $_{N->+\infty}$ (Angle) = 0 from the right

Limit $_{N->+\infty}$ (Apo) = the circle's radius (R) from the left

Limit $_{N->+\infty}$ (Perimeter) = the circle's circumference from the left

Limit $_{N->+\infty}$ (Angle) = the circle's area from the left

Geometrically, what is happening is that our regular polygons gets increasingly 'like the circle' as we increase the polygons number of sides.

Appendix: Common Classes and Scripts The following examples use the above classes and modules

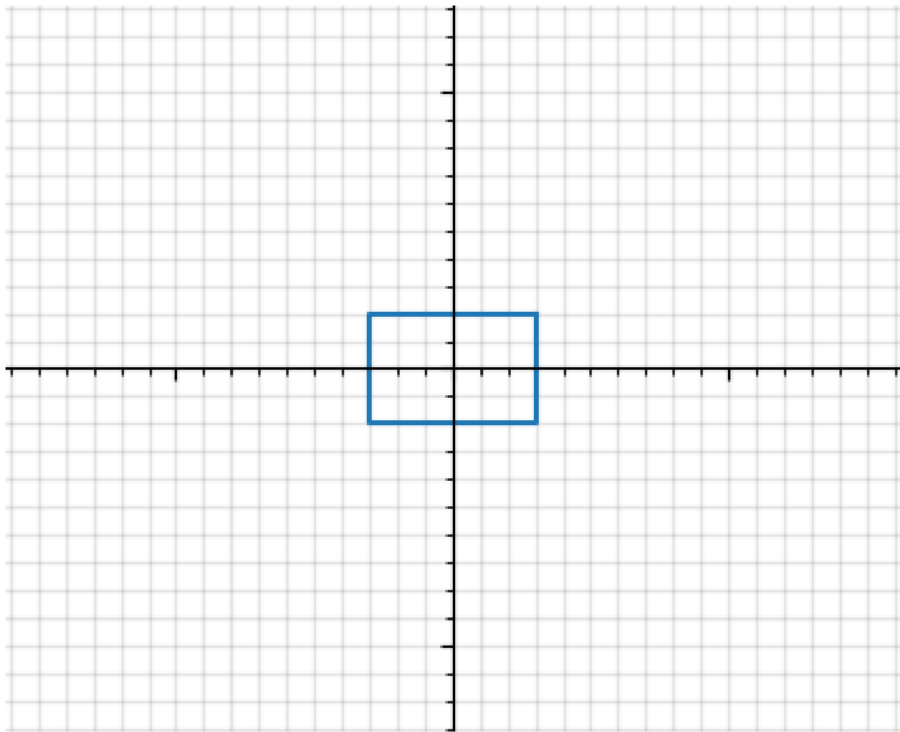
XY Coordinates

We now play!

Code: We plot a rectangle. Notice the **two** Python lines**: pointsA.append(Point(3, 2))** Try removing the **second** line

```
import matplotlib.pyplot as plt
import numpy as np
import XYUtils
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()
    points = Points()
    points.append(Point (3, 2))
    points.append(Point (-3, 2))
    points.append(Point (-3, -2))
    points.append(Point (3, -2))
    points.append(Point (3, 2))
    xyPlane.plot(points)
    plt.show()

# END
```

Output



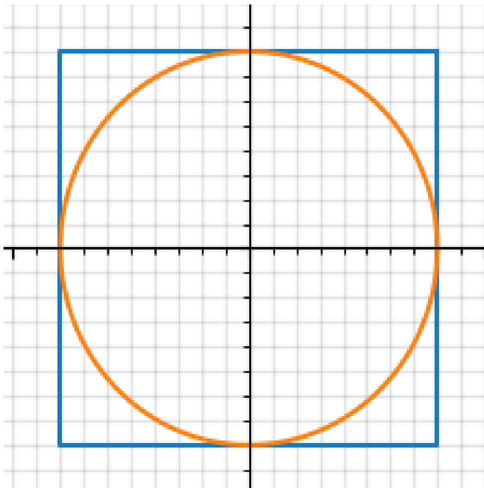
Code: We plot a circle inside a square. I just lied! We plot a regular polygon with a bunch of sides thar looks like a circle! We will discuss more about regular polygons is a subsequent chapter

Code:

```
import matplotlib.pyplot as plt
import math
from XYPlane import XYPlane
from Point import Point
from Points import Points


if __name__ == '__main__':
    xyPlane = XYPlane()
    squarePoints = Points()

    radius = 8

    squarePoints.append(Point (radius,  radius))
    squarePoints.append(Point (-radius, radius))
    squarePoints.append(Point (-radius, -radius))
    squarePoints.append(Point (radius, -radius))
    squarePoints.append(Point (radius, radius))
    xyPlane.plot(squarePoints)

    circum = 2.0 * math.pi
    step = circum / 2048.0

    rads = 0
    circlePoints = Points()
    while rads < circum:
        x = math.cos(rads) * radius
        y = math.sin(rads) * radius
        circlePoints.append(Point(x, y))
        rads = rads + step

    print("Regular polygon with " + str(circlePoints.size()) + " sides looks like a circle!")

    xyPlane.plot(circlePoints)
    plt.show()

# eof
```

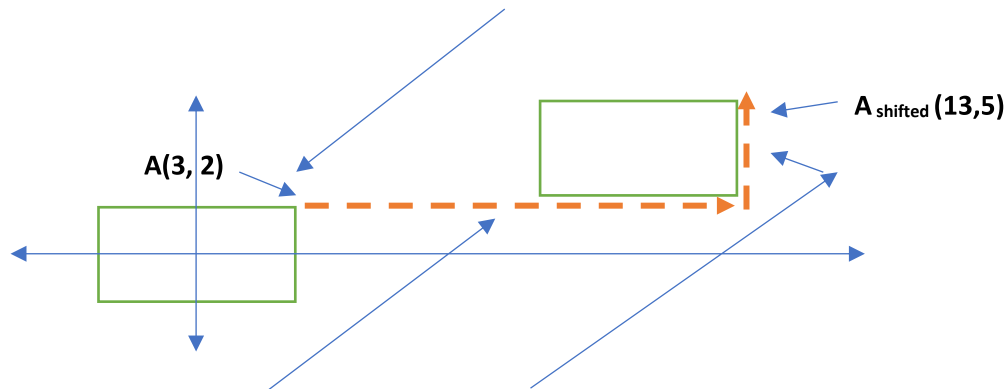
Output

Regular polygon with 2048 sides looks like a circle!



Polygon Shift

This is a polygon (here a rectangle) shift. Point A is a vertex point on our original rectangle (pre-shifted)



We shift horizontally 10 units and vertically 3 units; our xDelta is 10 and yDelta is 3

Code. We first plot our rectangle as before. We then **shift the rectangle's four vertex points and plot** the shifted points

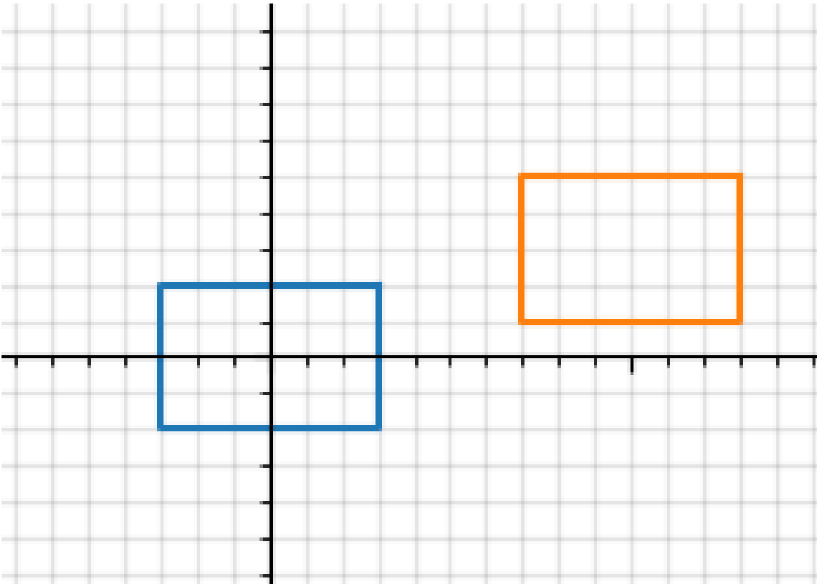
We exercise the method

xyShift(points: Points, xDelta, yDelta)

from out XYUtils.py module

```
import matplotlib.pyplot as plt
import numpy as np
import XYUtils
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()
    points = Points()
    points.append(Point(3, 2))
    points.append(Point(-3, 2))
    points.append(Point(-3, -2))
    points.append(Point(3, -2))
    points.append(Point(3, 2))
    xyPlane.plot(points)

    shiftedPoints = XYUtils.xyShift(points, 10, 3)
    xyPlane.plot(shiftedPoints)
    plt.show()

# END
```

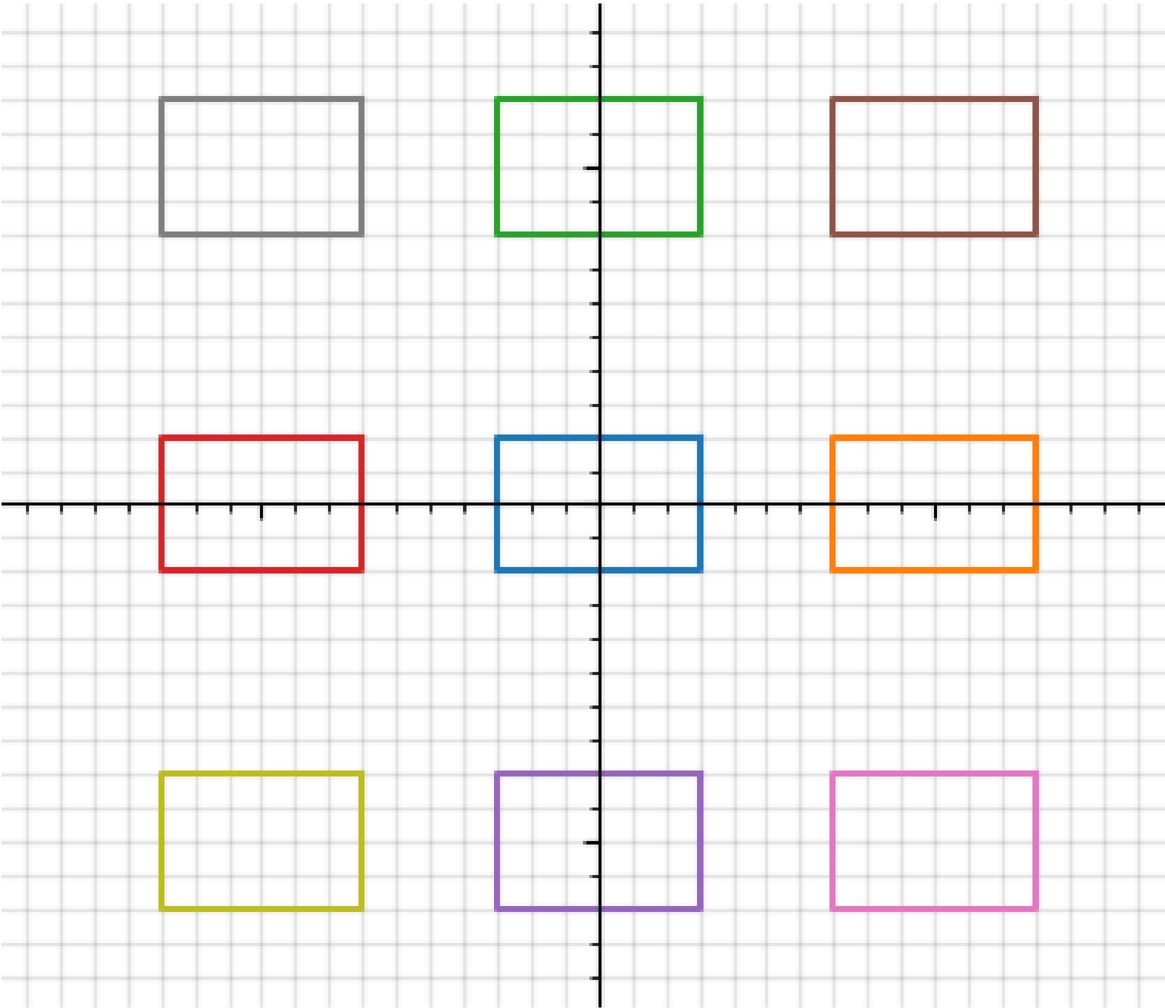
Output



Example: multiple shifts

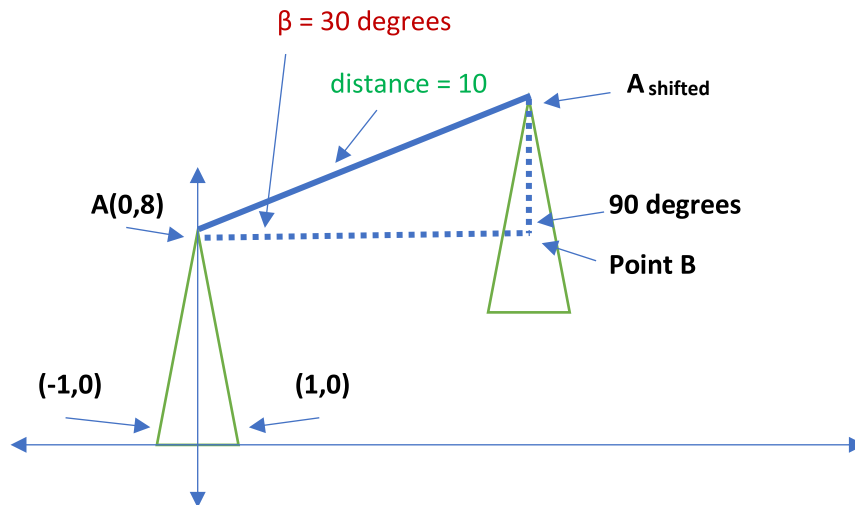
```
import matplotlib.pyplot as plt
import numpy as np
import XYUtils
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()
    points = Points()
    points.append(Point(3, 2))
    points.append(Point(-3, 2))
    points.append(Point(-3, -2))
    points.append(Point(3, -2))
    points.append(Point(3, 2))
    xyPlane.plot(points)

    shiftedPoints = XYUtils.xyShift(points, 10, 0)
    xyPlane.plot(shiftedPoints)

    shiftedPoints = XYUtils.xyShift(points, 0, 10)
    xyPlane.plot(shiftedPoints)

    shiftedPoints = XYUtils.xyShift(points, -10, 0)
    xyPlane.plot(shiftedPoints)

    shiftedPoints = XYUtils.xyShift(points, 0, -10)
    xyPlane.plot(shiftedPoints)

    shiftedPoints = XYUtils.xyShift(points, 10, 10)
    xyPlane.plot(shiftedPoints)

    shiftedPoints = XYUtils.xyShift(points, 10, -10)
    xyPlane.plot(shiftedPoints)

    shiftedPoints = XYUtils.xyShift(points, -10, 10)
    xyPlane.plot(shiftedPoints)

    shiftedPoints = XYUtils.xyShift(points, -10, -10)
    xyPlane.plot(shiftedPoints)

    plt.show()

# EOF
```

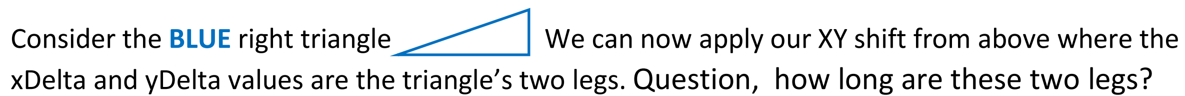
Output

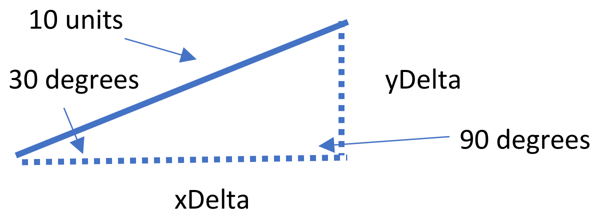


Example: triangle shift

Here is another shift. Point A is a vertex point on our isosceles triangle (pre-shifted). We shift the entire triangle 10 units via the angle 30 degrees β



Consider the **BLUE** right triangle  We can now apply our XY shift from above where the xDelta and yDelta values are the triangle's two legs. Question, how long are these two legs?



From our Trigonometry review in Volume One, we know:

sine(30) = opposite/hypotenuse = yDelta/10

cosine(30) = adjacent/hypotenuse = xDelta/10

Via my calculator

sine (30) = 0.5 = yDelta/10
yDelta = 5

cosine(30) = √3 /2 = xDelta/10
xDelta = 5 * √3 ≈ 8.6602540378443864676372317075294

We use ≈ to indicate 'approximately equal.' Note, our yDelta value is 'exactly' 5; our xDelta is approximately 8.6602540378443864676372317075294.

Triangle vertex points (-1, 0) (1, 0) (0, 8)
Apply xDelta and yDelta:
Shifted triangle points (7.66, 5) (9.66, 5) (8.66, 13)

More Shift methods

File: XYUtils.py

We now exercise two shift related methods in our Utils script:

- radianShift(points: Points, rads, distance)
- degreeShift(points: Points, degrees, distance)

We ‘code up’ our isosceles triangle example from above.

```
import matplotlib.pyplot as plt
import numpy as np
import math
import XYUtils
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()
    points = Points()
    points.append(Point(-1, 0))
    points.append(Point(1, 0))
    points.append(Point(0, 8))
    points.append(Point(-1, 0))
    xyPlane.plot(points)
    print("Baseline vertex points: " + points.str())

    distance = 10
    degreeShift = 30
    shiftedPoints = XYUtils.degreeShift(points, degreeShift, distance)
    xyPlane.plot(shiftedPoints)
    print("Shifted vertex points" + shiftedPoints.str())

    plt.show()

# eof
```

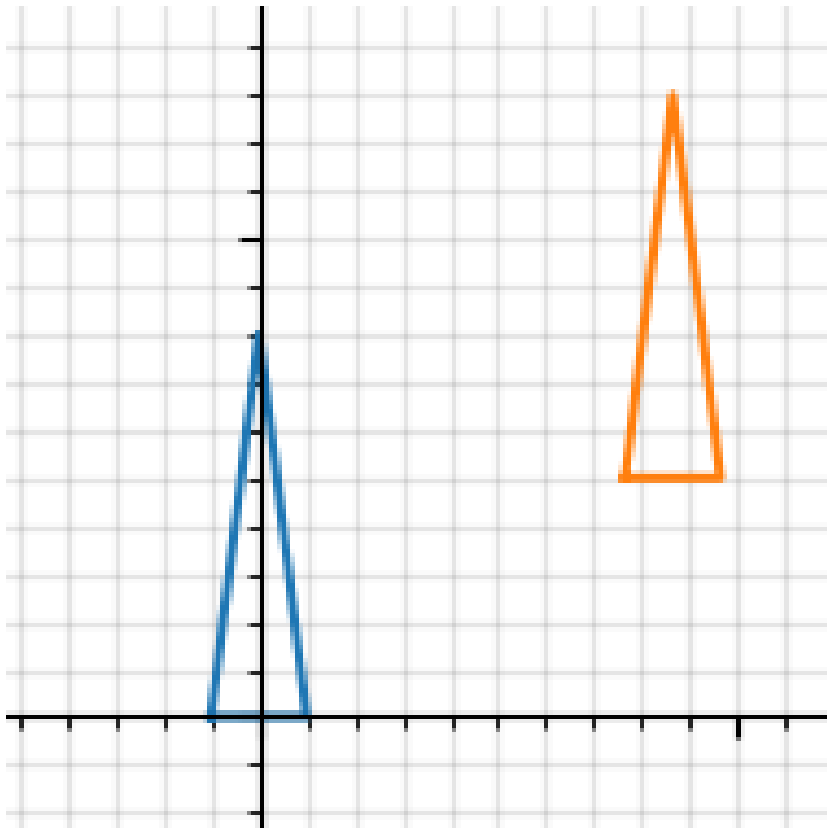
Expected results:

Triangle vertex points (-1,0) (1,0) (0, 8)
Shifter triangle points (7.66, 5) (9.66, 5) (8.66, 13)

Output

Baseline vertex points:
(-1,0)
(1,0)
(0,8)
(-1,0)

Shifted vertex points
(7.660254037844387,4.999999999999999)
(9.660254037844387,4.999999999999999)
(8.660254037844387,13.0)
(7.660254037844387,4.999999999999999)



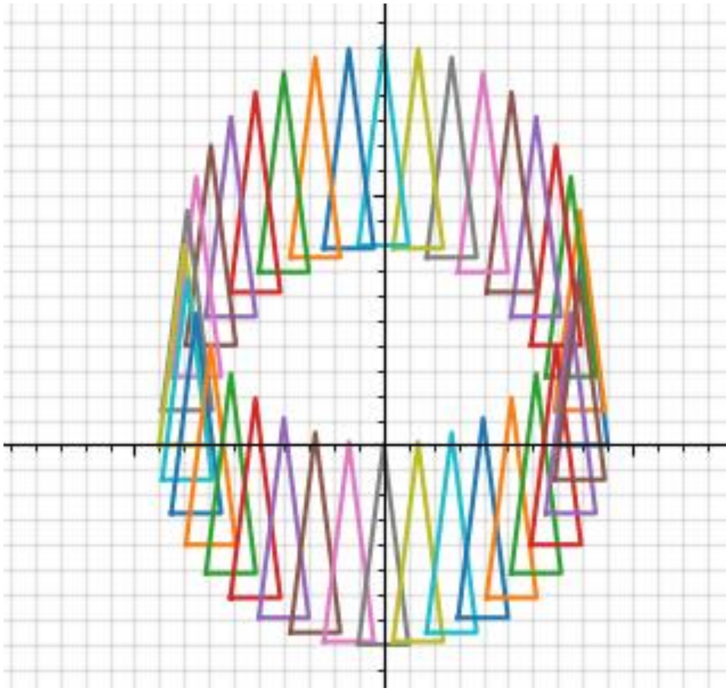
Multiple triangle shifts

Code:

```
import matplotlib.pyplot as plt
import numpy as np
import math
import XYUtils
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()
    points = Points()
    points.append(Point(-1, 0))
    points.append(Point(1, 0))
    points.append(Point(0, 8))
    points.append(Point(-1, 0))

    distance = 8
    degreeDelta = 10
    degreeShift = 0

    while degreeShift <= 360 - degreeDelta:
        shiftedPoints = XYUtils.degreeShift(points, degreeShift, distance)
        xyPlane.plot(shiftedPoints)
        degreeShift = degreeShift + degreeDelta

    plt.show()

# eof
```

Notice the original triangle is not plotted in this example

Output:

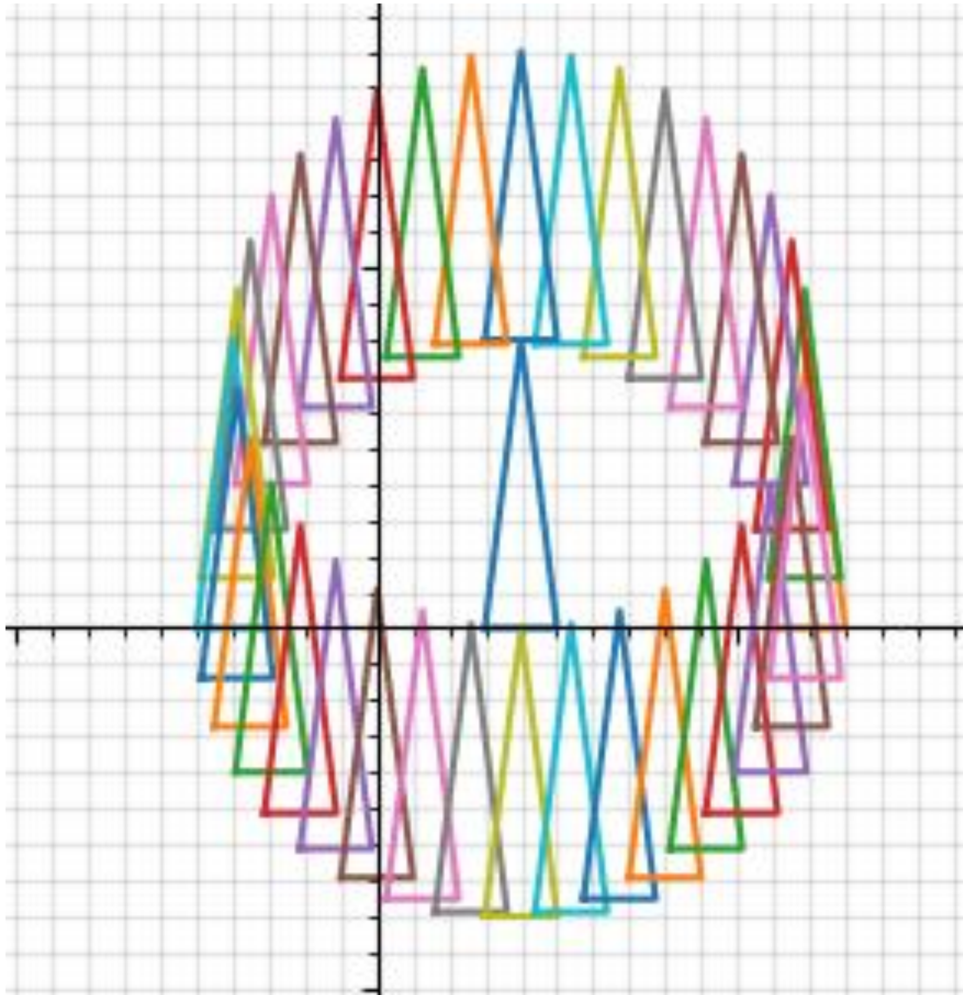


Multiple triangle shifts off center

Code:

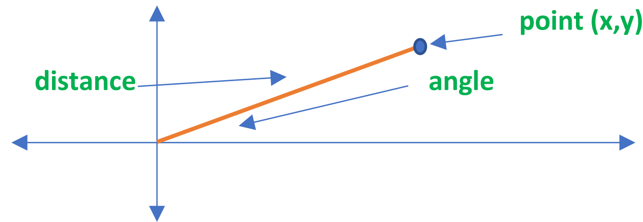
```
import matplotlib.pyplot as plt
import numpy as np
import math
import XYUtils
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()
    points = Points()
    points.append(Point(3, 0))
    points.append(Point(5, 0))
    points.append(Point(4, 8))
    points.append(Point(3, 0))
    xyPlane.plot(points)

    distance = 8
    degreeDelta = 10
    degreeShift = 0

    while degreeShift <= 360 - degreeDelta:
        shiftedPoints = XYUtils.degreeShift(points, degreeShift, distance)
        xyPlane.plot(shiftedPoints)
        degreeShift = degreeShift + degreeDelta

    plt.show()

# eof
```

Output:



Polar Coordinates

Review: in the Cartesian plane, a Polar coordinate **(distance, angle)** locates a **point** via an **angle** and a **distance** from the origin per the following diagram.



sine(**angle**) = **y/distance**
cosine(**angle**) = **x/ distance**
tangent(**angle**) = **y/x** where *x* != 0 (90 degree and 270-degree angles have NO tangent ratio)

Using the above trig ratios and taking into consideration the quadrant of the **point,** we can convert**:**

- XY Coordinate -> Polar Coordinate
- Polar Coordinate -> XY Coordinate

But first! - we look at Python's math arc tangent function. Consider 4 points (one per quadrant). We keep things by using a 45-degree angle and its friends - the 45, 135, 225 and 315 degree angles. The following four points are on the circle centered at (0,0) with radius R where R > 0

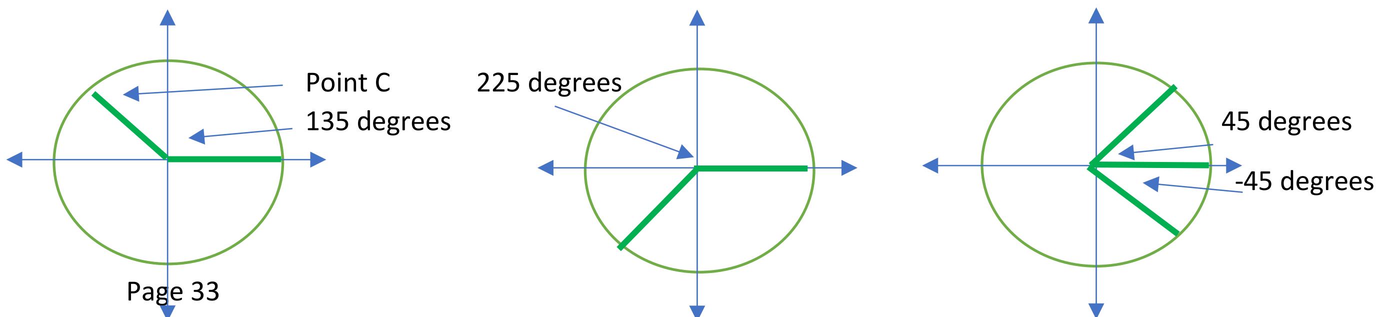
points	Quadrant	**xy** coordinates	Polar Coordinates (angle, distance)	tangent(**angle**) = **y/x**
A	1	(R, R)	(45, R)	R/R = 1
B	2	(-R, R)	(135, R)	R/-R = -1
C	3	(- R, - R)	(225, R)	-R/-R = 1
D	4	(R , - R)	(315, R)	-R/R = -1

Pythons math inverse tan

```
print("arcTan  -> " + str(math.degrees(math.atan(1))))
print("arcTan  -> " + str(math.degrees(math.atan(-1))))
```

Python outputs:
arcTan -> 45.0
arcTan -> -45.0

Consider the circle with radius R



The *arc tan function* takes one argument. It cannot distinguish among our four XY coordinate pairs. It does NOT take the xy coordinates as two arguments!

Angle	(X,Y) on circle	Quad	tangent(**angle**) = **y/x**	math arcTan (degrees) gives us these but….	We want (degrees)	We want corresponding Polar Coordinates (angle, distance)
45	(R, R)	1	tangent(45) = 1	arcTan(1) = 45	arc tan = 45	(45, R)
135	(-R, R)	2	tangent(135) = -1	arcTan(-1) = -45	arc tan = 135	(135, R)
225	(- R, - R)	3	tangent(225) = 1	arcTan(1) = 45	arc tan = 225	(225, R)
315	(R , - R)	4	tangent(315) = -1	arcTan(-1) = -45	arc tan = 315	(315, R)

The following logic gives us what we want

The following logic converts an XY coordinate to a Polar coordinate (we use degrees) This is a subset of our XYUtils.xy module

```
def __xyToPolarPointDegrees(p: Point):
    pp = PolarPoint()

    if p.x == 0 and p.y == 0:
        pp.distance = 0
        pp.angle = 0
        return pp

    if p.x == 0:
        pp.distance = abs(p.y)
        if p.y > 0:
            pp.angle = 90
        else:
            pp.angle = 270

        return pp

    if p.y == 0:
        pp.distance = abs(p.x)
        if p.x > 0:
            pp.angle = 0
        else:
            pp.angle = 180

        return pp

    #  else...
    pp.distance = __xround(math.sqrt(p.x * p.x + p.y * p.y))
    pp.angle = math.degrees(math.atan(p.y / p.x))
    if p.x < 0:
        pp.angle = 180 + pp.angle

    while pp.angle < 0.0:
        pp.angle = 360 + pp.angle

    pp.angle = __xround(pp.angle)
    return pp
```

Note: we adjust the angle returned by arcTan

We check for x coordinate < 0 (2nd and 3rd quadrants) and add 180 degrees in such cases

We check for negative valued angles and add 360 degrees to keep angles 0 <= angle < 360

Here is some of the test code that I used to verify coordinate conversions.

```
import matplotlib.pyplot as plt
import numpy as np
import math
import XYUtils
from XYPlane import XYPlane
from Point import Point
from PolarPoint import PolarPoint
from AngleMeasure import AngleMeasure
from Points import Points

if __name__ == '__main__':

    angleMeasure = AngleMeasure()
    angleMeasure.useDegrees()
    xyPlane = XYPlane()

    angles = [0, 30, 45, 90, 135, 180, 225, 270, 315, 360, 390]
    xyPoints = []
    i = 1
    for angle in angles:
        pp = PolarPoint()
        pp.angle = angle
        pp.distance = 8.0 * math.sqrt(2.0)
        p = XYUtils.polarPointToXY(pp, angleMeasure)
        xyPoints.append(p)
        print(str(i) + ") Polar -> " + pp.str() + "   XY -> " + p.str())
        plt.scatter(p.x, p.y)
        i = i + 1

    plt.show()

    i = 1
    for p in xyPoints:
        pp = XYUtils.xyToPolarPoint(p, angleMeasure)
        print(str(i) + ") XY -> " + p.str() + "   Polar -> " + pp.str())
        i = i + 1
# eof
```

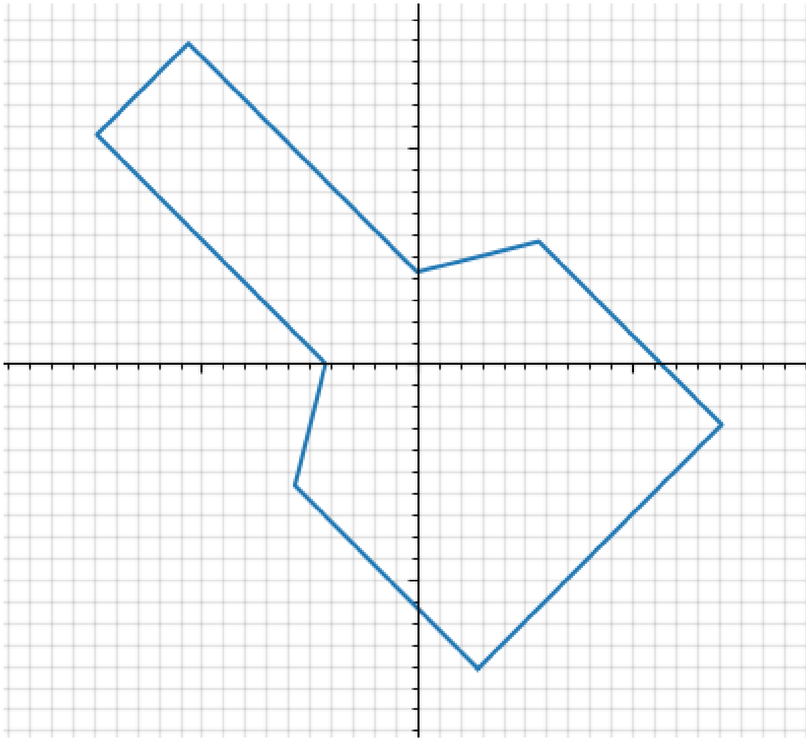
We now play

Wine bottle rotated

We rotate a wine bottle. Each rotation is in a separate XY Plane. We rotate 45 degrees for eight rotations

```
import matplotlib.pyplot as plt
import numpy as np
import math

import XYUtils
import PolarPoint
from AngleMeasure import AngleMeasure
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':

    angleMeasure = AngleMeasure()
    angleMeasure.useDegrees()

    xyPoints = Points()
    xyPoints.append(Point(-8, -12))
    xyPoints.append(Point(8, -12))
    xyPoints.append(Point(8, 0))
    xyPoints.append(Point(3, 3))
    xyPoints.append(Point(3, 18))
    xyPoints.append(Point(-3, 18))
    xyPoints.append(Point(-3, 3))
    xyPoints.append(Point(-8, 0))
    xyPoints.append(Point(-8, -12))

    xyPlane = XYPlane()
    xyPlane.plot(xyPoints)
    plt.show()

    pPointsList = []
    for xyPoint in xyPoints.asList():
        pPoint = XYUtils.xyToPolarPoint(xyPoint, angleMeasure)
        pPointsList.append(pPoint)

    rotates = 8
    rotateDegrees = 45
    rotateCount = 0
    while rotateCount < rotates:
        xyPoints.asList().clear()
        for pPoint in pPointsList:
            rotatedPoint = PolarPoint.rotate(pPoint, rotateDegrees, angleMeasure)
            pPoint.angle = rotatedPoint.angle
            xyPoint = XYUtils.polarPointToXY(pPoint, angleMeasure)
            xyPoints.append(xyPoint)
        xyPlane = XYPlane()
        xyPlane.plot(xyPoints)
        plt.show()
        rotateCount = rotateCount + 1
# eof
```

Here is the result of one of our eight rotations

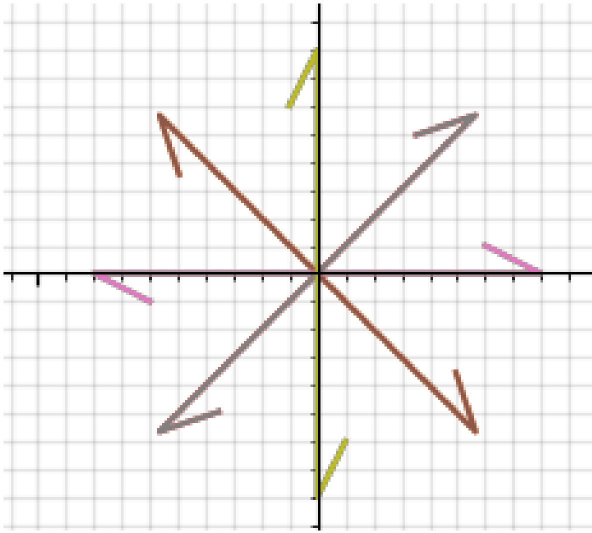


Rotations on one XY Plane

We rotate the following Shape

```
import matplotlib.pyplot as plt
import numpy as np
import math

import PolarPoint
import XYUtils
from AngleMeasure import AngleMeasure
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()

    angleMeasure = AngleMeasure()
    angleMeasure.useDegrees()

    xyPoints = Points()

    xyPoints.append(Point(-1, 6))
    xyPoints.append(Point(0, 8))
    xyPoints.append(Point(0, -8))
    xyPoints.append(Point(1, -6))

    xyPlane.plot(xyPoints)

    pPointsList = []
    for xyPoint in xyPoints.asList():
        pPoint = XYUtils.xyToPolarPoint(xyPoint, angleMeasure)
        pPointsList.append(pPoint)

    rotates = 7
    rotateDegrees = 45
    rotateCount = 0

    while rotateCount < rotates:
        xyPoints.asList().clear()

        for pPoint in pPointsList:
            rotatedPoint = PolarPoint.rotate(pPoint, rotateDegrees, angleMeasure)
            pPoint.angle = rotatedPoint.angle
            xyPoint = XYUtils.polarPointToXY(pPoint, angleMeasure)
            xyPoints.append(xyPoint)

        xyPlane.plot(xyPoints)
        rotateCount = rotateCount + 1

    plt.show()

# eof
```

Output

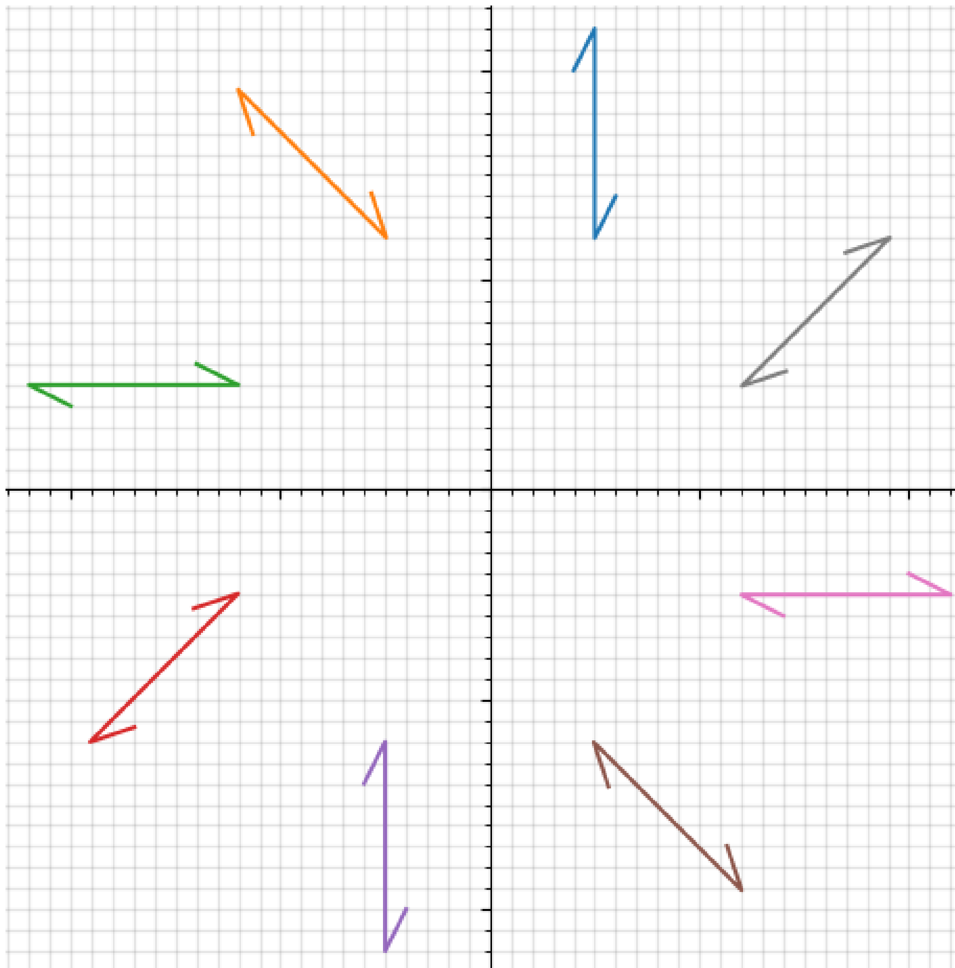


Shift and rotate example

We **shift once** and then **rotate seven times**

```
import matplotlib.pyplot as plt
import numpy as np
import math

import PolarPoint
import XYUtils
from AngleMeasure import AngleMeasure
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()

    angleMeasure = AngleMeasure()
    angleMeasure.useDegrees()

    tempPoints = Points()

    tempPoints.append(Point(-1, 11))
    tempPoints.append(Point(0, 13))
    tempPoints.append(Point(0, 3))
    tempPoints.append(Point(1, 5))

    xyPoints = XYUtils.xyShift(tempPoints, 5, 9)
    xyPlane.plot(xyPoints)

    pPointsList = []
    for xyPoint in xyPoints.asList():
        pPoint = XYUtils.xyToPolarPoint(xyPoint, angleMeasure)
        pPointsList.append(pPoint)


    rotates = 7
    rotateDegrees = 45

    rotateCount = 0
    while rotateCount < rotates:
        xyPoints.asList().clear()

        for pPoint in pPointsList:
            rotatedPoint = PolarPoint.rotate(pPoint, rotateDegrees, angleMeasure)
            pPoint.angle = rotatedPoint.angle
            xyPoint = XYUtils.polarPointToXY(pPoint, angleMeasure)
            xyPoints.append(xyPoint)

        xyPlane.plot(xyPoints)
        rotateCount = rotateCount + 1

    plt.show()

# eof
```

Output

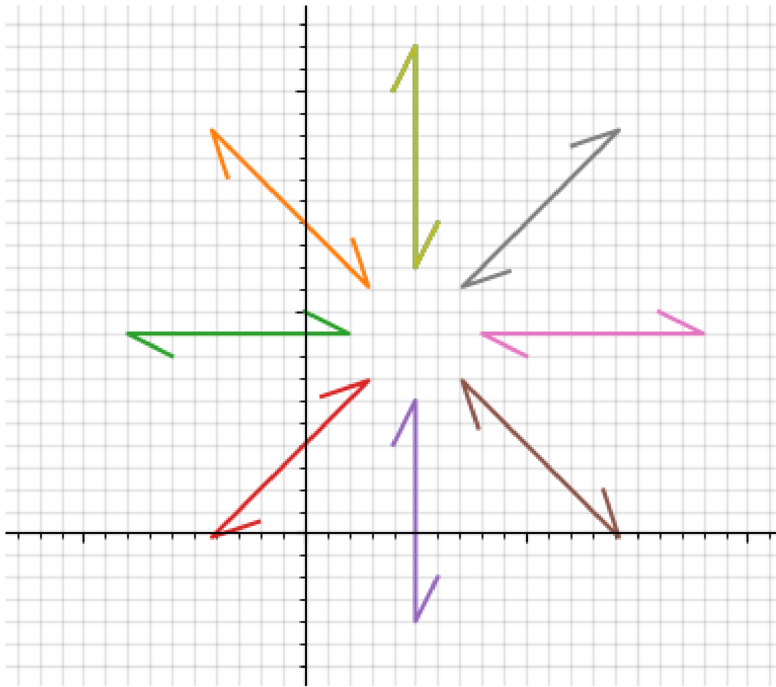


Rotate and shift example

In our while **loop**, we **rotate** and then **shift**

```
import matplotlib.pyplot as plt
import numpy as np
import math

import PolarPoint
import XYUtils
from AngleMeasure import AngleMeasure
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()

    angleMeasure = AngleMeasure()
    angleMeasure.useDegrees()

    xyPoints = Points()

    xyPoints.append(Point(-1, 11))
    xyPoints.append(Point(0, 13))
    xyPoints.append(Point(0, 3))
    xyPoints.append(Point(1, 5))

    xyPlane.plot(XYUtils.xyShift(xyPoints, 5, 9))

    pPointsList = []
    for xyPoint in xyPoints.asList():
        pPoint = XYUtils.xyToPolarPoint(xyPoint, angleMeasure)
        pPointsList.append(pPoint)

    rotates = 8
    rotateDegrees = 45

    rotateCount = 0
    while rotateCount < rotates:
        xyPoints.asList().clear()

        for pPoint in pPointsList:
            rotatedPoint = PolarPoint.rotate(pPoint, rotateDegrees, angleMeasure)
            pPoint.angle = rotatedPoint.angle
            xyPoint = XYUtils.polarPointToXY(pPoint, angleMeasure)
            xyPoints.append(xyPoint)

        xyPlane.plot(XYUtils.xyShift(xyPoints, 5, 9))
        rotateCount = rotateCount + 1

    plt.show()

# eof
```

Output

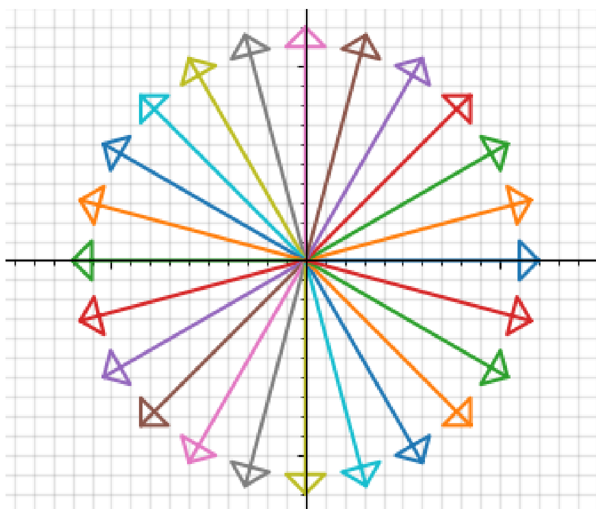


Spin an arrow

We spin (rotate) an arrow!

```
import matplotlib.pyplot as plt
from AngleMeasure import AngleMeasure
from Spinner import XYDegreeSpinner
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()

    angleMeasure = AngleMeasure()
    angleMeasure.useDegrees()

    xyPoints = Points()

    xyPoints.append(Point(0,0))
    xyPoints.append(Point(12, 0))
    xyPoints.append(Point(11, 1))
    xyPoints.append(Point(11, -1))
    xyPoints.append(Point(12,0))

    spinCount = 0
    degrees = 15
    spins = 24

    spinner = XYDegreeSpinner(xyPoints, degrees)
    while spinCount < spins:
        xyPlane.plot(xyPoints)
        xyPoints = spinner.nextRotation()
        spinCount = spinCount + 1;

    plt.show()

# eof
```

Output

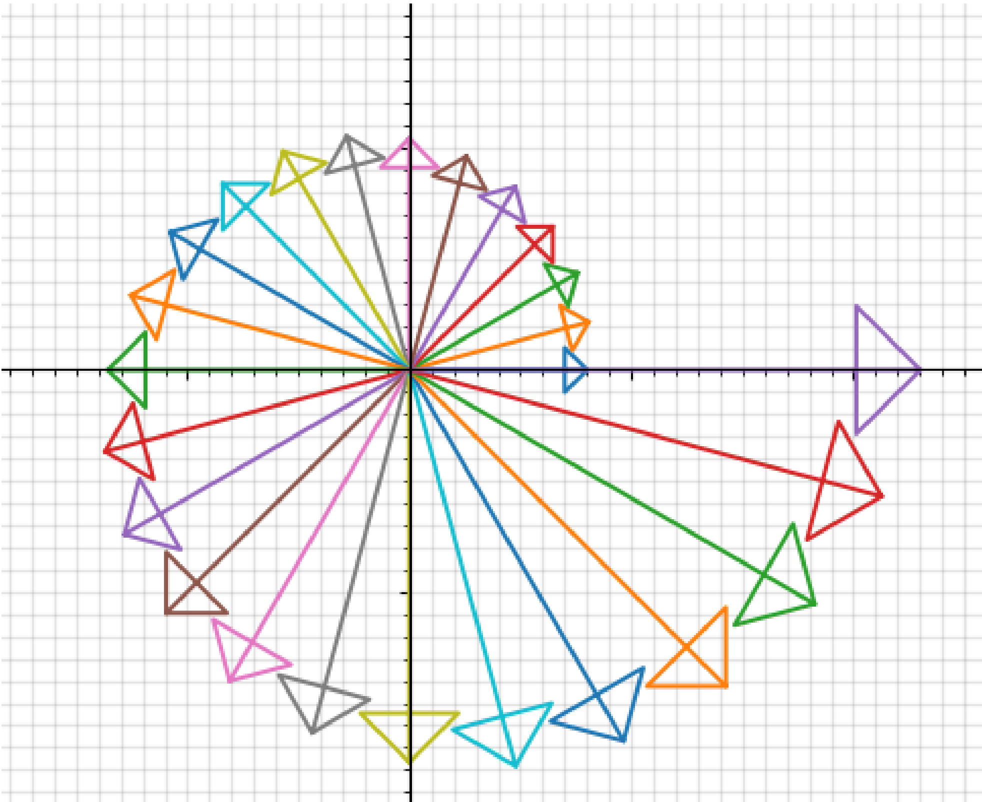


Spin and enlarge an arrow

We spin (rotate) an arrow! We **change distance (arrow length) for each rotation**

```
import matplotlib.pyplot as plt
import PolarPoint
import XYUtils
from AngleMeasure import AngleMeasure
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':

    angleMeasure = AngleMeasure()
    angleMeasure.useDegrees()
    distance = 8
    xyPoints = Points()
    xyPoints.append(Point(0, 0))
    xyPoints.append(Point(distance, 0))
    xyPoints.append(Point(distance-1, 1))
    xyPoints.append(Point(distance-1, -1))
    xyPoints.append(Point(distance, 0))

    xyPlane = XYPlane()
    xyPlane.plot(xyPoints)

    pPointsList = []
    for xyPoint in xyPoints.asList():
        pPoint = XYUtils.xyToPolarPoint(xyPoint, angleMeasure)
        pPointsList.append(pPoint)

    rotates = 24
    rotateDegrees = 15
    rotateCount = 0

    while rotateCount < rotates:
        xyPoints.asList().clear()

        for pPoint in pPointsList:
            a = PolarPoint.incrementAngle(pPoint.angle, rotateDegrees, angleMeasure)
            pPoint.angle = a
            pPoint.distance = pPoint.distance*1.045

            xyPoint = XYUtils.polarPointToXY(pPoint, angleMeasure)
            xyPoints.append(xyPoint)


        xyPlane.plot(xyPoints)

        rotateCount = rotateCount + 1

    plt.show()

# eof
```

Output



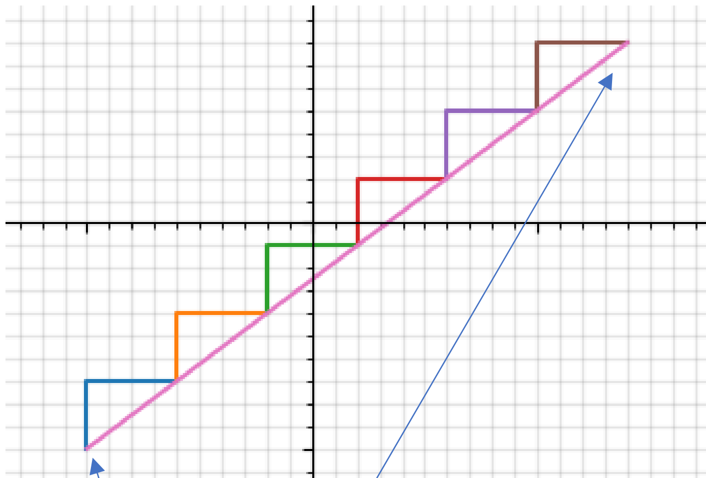
Stairway to nowhere

We build stairs

Code

```
import matplotlib.pyplot as plt
import XYUtils
from XYPlane import XYPlane
from Point import Point
from Points import Points

if __name__ == '__main__':
    xyPlane = XYPlane()
    points = Points()
    firstPoint = Point(-10, -10)
    points.append(firstPoint)
    points.append(Point(-10, -7))
    points.append(Point(-6, -7))

    xyPlane.plot(points)

    J = 0
    deltaX = 4
    deltaY = 3

    deltaXSum = deltaX
    deltaYSum = deltaY

    while J < 5:
        points = XYUtils.xyShift(points, deltaX, deltaY)
        deltaXSum = deltaXSum + deltaX
        deltaYSum = deltaYSum + deltaY
        xyPlane.plot(points)
        J = J + 1

    lastPoint = points.asList()[-1]
    firstLast = Points()
    firstLast.append(firstPoint)
    firstLast.append(lastPoint)
    xyPlane.plot(firstLast)

    d = XYUtils.distance(lastPoint, firstPoint)
    print("First point: " + firstPoint.str() + " Last point: " + lastPoint.str())
    print("Delta X sum: " + str(deltaXSum) + " Delta Y sum: " + str(deltaYSum) + " Hypotenuse:
" + str(d))

    plt.show()

# eof
```

Output



First point: (-10,-10) Last point: (14,8)
Delta X sum: 24 Delta Y sum: 18 Hypotenuse: 30.0

Notes:

Each of our steps is a 3-4-5 Pythagorean triplet because our x delta is 4 and y delta is 3!

The distance (from our first point to our last point is 30

18, 24, 30 form a right triangle similar to our 3,4,5, right triangles. Pythagorean theorem

$18^2 + 24^2 = 30^2$
324 + 576 = 900

We have 6 steps. Remember our ratio discussion re similar triangles in Volume One!

18 / 6 = 3   24/6 =4   30/6 = 5

A Circle

Regular Polygon Builder

We begin by plotting a regular polygon centered at the origin.

Definition: A polygon is regular when all angles are equal, ***and*** all sides are equal

We jump right into the code.

We code a regular polygon builder. The builder requires two inputs

- The regular polygon's circumradius
- The number of sides N where N >= 3

To keep our examples small enough to plot we use 8 for our circumradius.

Code with default values. Needed to build a regular polygon

```
class RegularPolygonBuilderPacket:
    def __new__(cls, *args, **kwargs):
        return super().__new__(cls)

    def __init__(self):
        self.circumRadius = 8
        self.numSides = 3
# END
```

Regular polygon builder

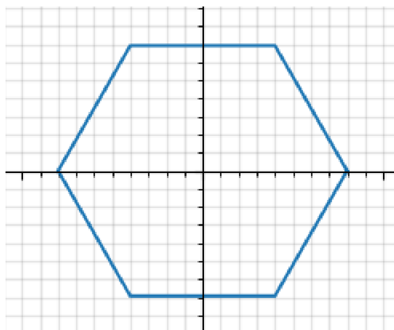
```
import math
import XYUtils
from AngleMeasure import AngleMeasure
from PolarPoint import PolarPoint
from Points import Points
from RegularPolygonBuilderPacket import RegularPolygonBuilderPacket


class RegularPolygonBuilder:
    def __new__(cls, *args, **kwargs):
        return super().__new__(cls)

    def __init__(self, buildPacket: RegularPolygonBuilderPacket):
        self.circumRadius = buildPacket.circumRadius
        self.numSides = buildPacket.numSides
        self.side = 0
        self.apothem = 0
        self.area = 0
        self.perimeter = 0
        self.angle = 0
        self.xyPoints = Points()
        self.angle = 360.0 / self.numSides

        angleMeasure = AngleMeasure()
        angleMeasure.useDegrees()

        ppAngle = 0
        while ppAngle <= 360:
            pPoint = PolarPoint()
            pPoint.distance = self.circumRadius
            pPoint.angle = ppAngle
            xyPoint = XYUtils.polarPointToXY(pPoint, angleMeasure)
            self.xyPoints.append(xyPoint)
            ppAngle = ppAngle + self.angle

        xyList = self.xyPoints.asList()
        p1 = xyList[0]
        p2 = xyList[1]
        self.side = XYUtils.distance(p1, p2)

        self.perimeter = self.numSides * self.side
        self.apothem = math.sqrt(self.circumRadius*self.circumRadius-self.side/2*self.side/2)
        self.area = self.perimeter * self.apothem / 2.0

# END
```

We now exercise the builder. We build a **Hexagon - a six-sided** regular polygon. Change the 6 to other integer values to play!

```
import matplotlib.pyplot as plt
from RegularPolygonBuilder import RegularPolygonBuilder
from RegularPolygonBuilderPacket import RegularPolygonBuilderPacket
from XYPlane import XYPlane

if __name__ == '__main__':
    xyPlane = XYPlane()

    builderPacket = RegularPolygonBuilderPacket()
    builderPacket.numSides = 6
    builderPacket.circumRadius = 8

    builder = RegularPolygonBuilder(builderPacket)

    print("Polygon sides : " + str(builder.numSides) + " Side length: " + str(builder.side))
    print("Polygon perimeter : " + str(builder.perimeter))
    print("Polygon area : " + str(builder.area))
    print("Polygon apothem : " + str(builder.apothem))
    print("Polygon angle : " + str(builder.angle))

    xyPlane.plot(builder.xyPoints)
    plt.show()

# END
```

Output

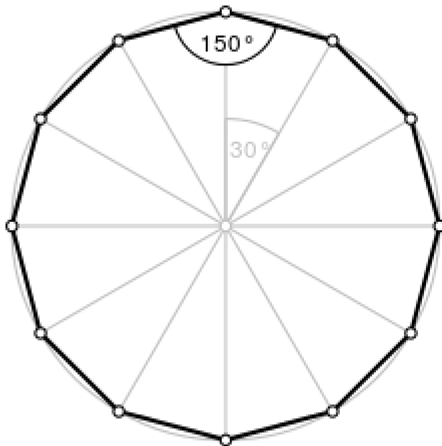
Polygon sides : 6   Side length: 7.999997202499511
Polygon perimeter : 47.999983214997066
Polygon area : 166.27683876329706
Polygon apothem : 6.928204037844151
Polygon angle : 60.0



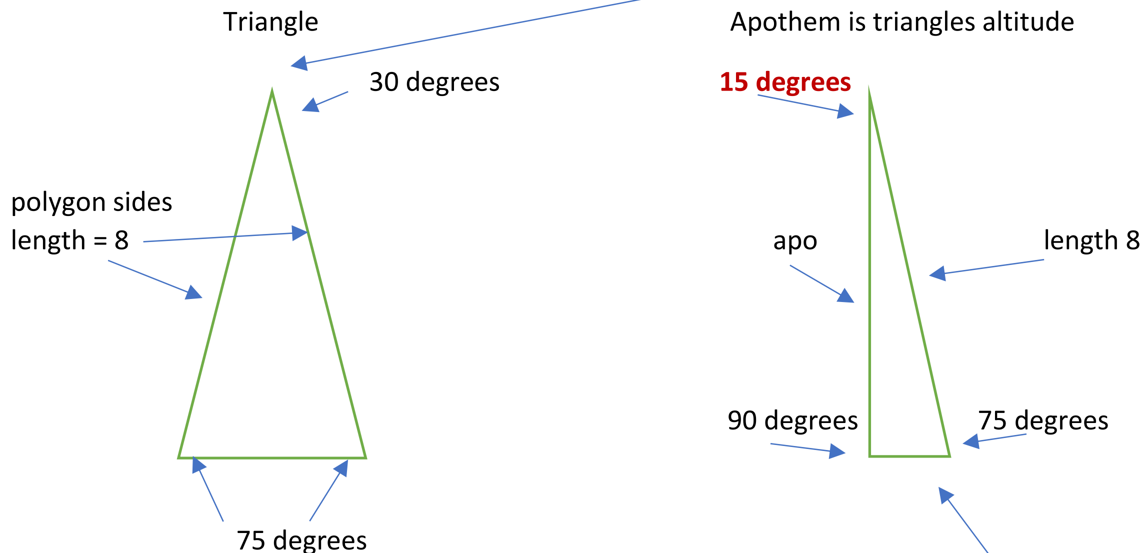
Here is the polygon stats with 12 sides

Polygon sides : 12 Side length: 4.141105557698331
Polygon perimeter : 49.693266692379964
Polygon area : 192.00003598027567
Polygon apothem : 7.727406498302002
Polygon angle : 30.0

I stole this diagram from the web!



We now check the python math in our builder. We inspect one of the above isosceles triangles (they are all congruent)



The apothem forms two right triangles from our isosceles triangle. We display one of the two. Consider the **15-degree angle above** (you can also use the 75-degree angle!) Because the triangle is a right triangle we can use our calculator to calculate the following.

sine(15) = 0.25881904510252076234889883762405 = opposite/ hypotenuse
0.25881904510252076234889883762405 = opposite/8
2.0705523608201660987911907009924 = opposite

cosine(15) = 0.9659258262890682867497431997289 = adjacent/hypotenuse
0.9659258262890682867497431997289 = adjacent/8
7.7274066103125462939979455978312 = adjacent

Looks closely, the triangle opposite side is ½ the polygon side
Polygon side = 2.0705523608201660987911907009924 * 2
Polygon side = 4.1411047216403321975823814019848

The triangle adjacent side is the polygon apothem
Polygon apothem = 7.7274066103125462939979455978312

Our polygon has 12 sides

Polygon perimeter = 12 * 4.1411047216403321975823814019848
Polygon perimeter = 49.693256659683986370988576823817

In volume One we determined the formula for area of a regular polygon
area = perimeter * apothem /2

Polygon Area = 49.693256659683986370988576823817 * 7.7274066103125462939979455978312/ 2
Polygon area = 192

	My calculator	**My Python output**
side	4.1411047216403321975823814019848	4.141105557698331
perimeter	49.693256659683986370988576823817	49.693266692379964
area	192	192.00003598027567
apo	7.7274066103125462939979455978312	7.727406498302002

Math Sequences

We take a bit of a detour here. And talk about sequences. A sequence of numbers is an ordered list of numbers. The sequence usually has a pattern. Because the sequence is ordered we can list them and count them!

Example sequence A

Sequence : { 5, 10, 15, 20, 25…}

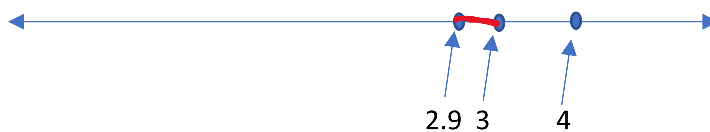
I ended our list with a … since we all can see the simple pattern - at least I hope we can!

Notice that as we move thru this sequence from left to right the element numbers get bigger and bigger!

Example sequence B

Sequence (2.9, 2.99, 2.999, 2.9999…}

Notice that as we move thru this sequence from left to right the element numbers get bigger and bigger! For example, it is correct to say that as we move from left to right → the elements get closer and closer to 4

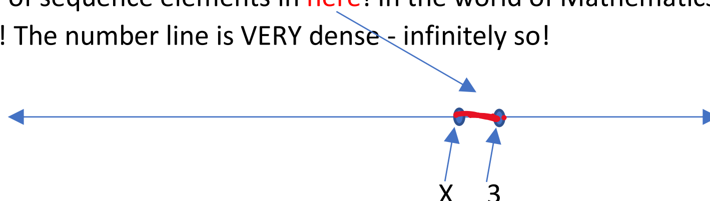


The elements of our sequence move along the RED segment of our number line in a left to right direction so that they get closer and closer to 4. They also get closer and closer **to 5 and 3 and 6 and 10.9** and an infinite number of other numbers! They do NOT move closer to 1. In fact, they move farther and farther away from 1!

We looked **at 4 and 3 and 5 and 6 and 10.9** above. Now consider the infinite set of numbers consisting of ALL the numbers that our sequence is moving closer to!

Question what is the smallest number in our set? Excuse me for asking an 'easy question'! Of course, our answer is 3!

Pick any number X 'really close' to 3 on its left side (X < 3) X = 2.99999999992999919191919 will do. As our sequence moves along from left to right it will eventually 'pass' X; in fact, there are an infinite number of sequence elements in here! In the world of Mathematics, the word 'close' is extremely relative! The number line is VERY dense - infinitely so!



Our sequence B is said to ***converge to*** 3. Specifically, it ***converges towards 3 from the left***; every sequence element is < 3. The number 3 is NOT an element of our sequence

We can use the notation S_N for the Nth element in our sequence, e.g., $S_4 = 2.9999$
Now we use the following conventional notation for the convergence. We say that **sequence B** ***approaches 3 from the left***

$$\text{Limit}_{N \to +\infty} (S_N) = 3$$

We say that **sequence A** ***approaches positive infinity***

$$\text{Limit}_{N \to +\infty} (S_N) = +\infty$$

Another sequence example

$S_N = \{1/10^1, 1/10^2, 1/10^3, 1/10^4, \ldots\}$ here $SN = 1/10^N$

This sequence ***approaches 0 from the right***

$$\text{Limit}_{N \to +\infty} (S_N) = 0$$

A circle

We take the above Python code and remove the point plotting

```
import matplotlib.pyplot as plt
from RegularPolygonBuilder import RegularPolygonBuilder
from RegularPolygonBuilderPacket import RegularPolygonBuilderPacket
from XYPlane import XYPlane

if __name__ == '__main__':
    xyPlane = XYPlane()

    builderPacket = RegularPolygonBuilderPacket()
    builderPacket.numSides = 64
    builderPacket.circumRadius = 8

    builder = RegularPolygonBuilder(builderPacket)

    print("Polygon sides : " + str(builder.numSides) + " Side length: " + str(builder.side))
    print("Polygon perimeter : " + str(builder.perimeter))
    print("Polygon area : " + str(builder.area))
    print("Polygon apothem : " + str(builder.apothem))
    print("Polygon angle : " + str(builder.angle))

   # xyPlane.plot(builder.xyPoints)
   #  plt.show()

# END
```

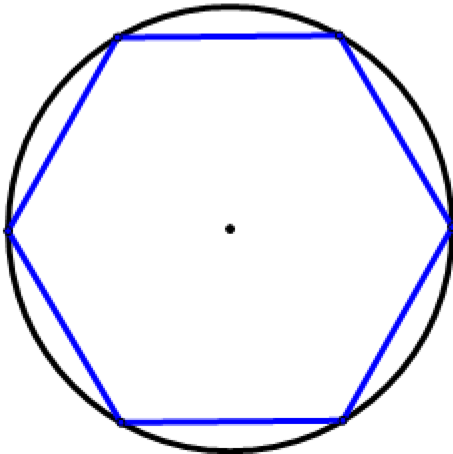
I changed my rounding in XYUtils.py

```
ROUND_DECIMALS = 16
```

We perform multiple runs and get the following outputs

Sides N	Angle (degrees)	Apo	Perimeter	Area
64	5.625	7.990363649641379	50.24529851127605	200.73910339494012
128	2.8125	7.997590549569634	50.260436014924366	200.9811940451042
256	1.40625	7.999397614713156	50.26422081830882	201.04174405969746
512	0.703125	7.999849402260809	50.26516704587346	201.05688327323526
1024	0.3515625	7.9999623504766095	50.26540360443455	201.06066818349385
2048	0.17578125	7.999990587613615	50.265462744179175	201.06161441773813
4096	0.087890625	7.999997646903058	50.26547752912178	201.06185097671639
8192	0.0439453125	7.999999411725743	50.26548122535767	201.06191011648636

If we inscribe any one of the above regular polygons we can visually compare them to the surrounding circle of Radius 8. Our polygons have a bunch more sides than our diagram below BUT you should get the picture!



The surrounding circle has a radius of 8

```
builderPacket.circumRadius = 8
```

Circle details:

circle circumference = Pi * diameter
circle circumference = Pi * 16

circle circumference ≈ 3.14159265358979323846264338327950 * 16

circle circumference ≈ 50.2654824574366918

circle area = Pi * radius * radius

circle area ≈ 3.14159265358979323846264338327955 * 64

circle area ≈ 201.06192982974676726161

Here is our polygon table from above

Sides N	Angle (degrees)	Apo	Perimeter	Area
64	5.625	7.990363649641379	50.24529851127605	200.73910339494012
128	2.8125	7.997590549569634	50.260436014924366	200.9811940451042
256	1.40625	7.999397614713156	50.26422081830882	201.04174405969746
512	0.703125	7.999849402260809	50.26516704587346	201.05688327323526
1024	0.3515625	7.9999623504766095	50.26540360443455	201.06066818349385
2048	0.17578125	7.999990587613615	50.265462744179175	201.06161441773813
4096	0.087890625	7.999997646903058	50.26547752912178	201.06185097671639
8192	0.0439453125	7.999999411725743	50.26548122535767	201.06191011648636
Our surrounding circle				
		Radius	Circumference	Area
		8	50.26548245743669	201.0619298297467672

We can treat each green column in our table as a sequence of numbers.

The apothems are getting larger and larger and are all < 8 (the circle radius)

The perimeters are getting larger and larger and are all < circle circumference

The areas are getting larger and larger and are all < circle area

Using the 'limit' notation we see:

Limit $_{N->+\infty}$ (Angle) = 0 from the right

Limit $_{N->+\infty}$ (Apo) = the circle's radius (R) from the left

Limit $_{N->+\infty}$ (Perimeter) = the circle's circumference from the left

Limit $_{N->+\infty}$ (Angle) = the circle's area from the left

Geometrically, what is happening is that our regular polygons gets increasingly 'like the circle' as we increase the polygons number of sides.

Appendix: Common Classes and Scripts

Class Point

File: Point.py

An XY coordinate pair

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

    def str(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"
# END
```

Class Points

File: Points.py

Here we maintain a list of points. We maintain the ordering sequence as our user appends points

```
from Point import Point


class Points:
    def __new__(cls, *args, **kwargs):
        return super().__new__(cls)

    def __init__(self):
        self.points = []

    def append(self, point: Point):
        self.points.append(point)

    def asList(self):
        return self.points

    def getXs(self):
        Xs = []
        for point in self.points:
            Xs.append(point.x)
        return Xs

    def getYs(self):
        Ys = []
        for point in self.points:
            Ys.append(point.y)
        return Ys

    def str(self):
        s = ""
        for point in self.points:
            s = s + point.str() + " "
        return s

    def size(self):
        return len(self.points)

# END
```

Class XY Plane

File: XYPlane.py

We mimic graph paper with highlighted X and Y axes. We supply one public method to **plot** Points.

```
import numpy as np
import matplotlib.pyplot as plt
import Point
import Points


class XYPlane:
    def __new__(cls, *args, **kwargs):
        return super().__new__(cls)

    def __init__(self):
        fig, self.ax = plt.subplots(figsize=(10, 10))
        # Select length of axes and the space between tick labels
        unit = 25
        xMin, xMax, yMin, yMax = -unit, unit, -unit, unit
        ticks_frequency = 1
        # Set identical scales for both axes
        self.ax.set(xlim=(xMin - 1, xMax + 1), ylim=(yMin - 1, yMax + 1), aspect='equal')

        # Set bottom and left spines as x and y axes of coordinate system
        self.ax.spines['bottom'].set_position('zero')
        self.ax.spines['left'].set_position('zero')

        # Remove top and right spines
        self.ax.spines['top'].set_visible(False)
        self.ax.spines['right'].set_visible(False)

        self.ax.set_xticks(np.arange(xMin, xMax + 1), minor=True)
        self.ax.set_yticks(np.arange(yMin, yMax + 1), minor=True)
        self.ax.set_xticklabels('');
        self.ax.set_yticklabels('');

        # Draw grid lines - like graph paper
        self.ax.grid(which='both', color='grey', linewidth=1, linestyle='-', alpha=0.2)

    def plot(self, points: Points):
        self.ax.plot(points.getXs(), points.getYs())

# EOF
```

Class AngleMeasure

File: AngleMeasure.py

We distinguish between degrees and radians

```
class AngleMeasure:
    # default to Radians
    def __init__(self):
        self.unit = AngleMeasure.RADIANS

    RADIANS = "Radians"
    DEGREES = "Degrees"

    def useRadians(self):
        self.unit = AngleMeasure.RADIANS

    def useDegrees(self):
        self.unit = AngleMeasure.DEGREES
# END
```

Class PolarPoint

File: PolarPoint.py

```
import math

from AngleMeasure import AngleMeasure


class PolarPoint:
    def __init__(self):
        self.angle = 0.0
        self.distance = 0.0

    def str(self):
        return "(" + str(self.distance) + ", " + str(self.angle) + ")"


def rotate(pp: PolarPoint, rotation, angleMeasure: AngleMeasure):
    rotatedPoint = PolarPoint()
    rotatedPoint.distance = pp.distance;
    rotatedPoint.angle = incrementAngle(pp.angle, rotation, angleMeasure)

    # print("XYUtils Rotate : point: " + pp.str() + " rotation: " + str(rotation)
    #      + " rotated point: " + rotatedPoint.str())

    return rotatedPoint


def incrementAngle(angle, rotation, angleMeasure: AngleMeasure):
    incrementedAngle = angle + rotation

    adjust = 360.0
    if angleMeasure.unit == AngleMeasure.RADIANS:
        adjust = math.radians(360.0)

    while incrementedAngle >= adjust:
        incrementedAngle = incrementedAngle - adjust

    while incrementedAngle < 0:
        incrementedAngle = incrementedAngle + adjust

    return incrementedAngle

# END
```

Module XY Utils

File: XYUtils.py

```
import numpy as np
import matplotlib.pyplot as plt
import math
from Point import Point
from AngleMeasure import AngleMeasure
from PolarPoint import PolarPoint
from Points import Points

ROUND_DECIMALS = 5


def __xround(x):
    return round(x, ROUND_DECIMALS)


# to avoid Python's annoying -0.0
def __xsin(r):
    s = math.sin(r)
    if __xround(s) == 0.0:
        s = abs(s)
    return s


# to avoid Python's annoying -0.0
def __xcos(r):
    s = math.cos(r)
    if __xround(s) == 0.0:
        s = abs(s)
    return s


def xyShift(points: Points, xDelta, yDelta):
    shiftedPoints = Points()
    for point in points.asList():
        p = Point(point.x + xDelta, point.y + yDelta)
        shiftedPoints.append(p)
    return shiftedPoints


def radianShift(points: Points, rads, distance):
    xDelta = __xround(__xcos(rads) * distance)
    yDelta = __xround(__xsin(rads) * distance)
    return xyShift(points, xDelta, yDelta)


def degreeShift(points: Points, degrees, distance):
    rads = math.radians(degrees)
    return radianShift(points, rads, distance)


def distance(p1: Point, p2: Point):
    d = math.sqrt((p1.y - p2.y) * (p1.y - p2.y) + (p1.x - p2.x) * (p1.x - p2.x))
    return d
```

```
def polarPointToXY(pp: PolarPoint, angleMeasure: AngleMeasure):
    angle = pp.angle

    if angleMeasure.unit == AngleMeasure.DEGREES:
        angle = math.radians(pp.angle)

    x = __xround(pp.distance * __xcos(angle))
    y = __xround(pp.distance * __xsin(angle))

    return Point(x, y)


def xyToPolarPoint(p: Point, angleMeasure: AngleMeasure):
    pp = __xyToPolarPointDegrees(p)

    if angleMeasure.unit == AngleMeasure.RADIANS:
        pp.angle = math.radians(pp.angle)
    return pp


# private method -- here, we use degrees
def __xyToPolarPointDegrees(p: Point):
    pp = PolarPoint()

    if p.x == 0 and p.y == 0:
        pp.distance = 0
        pp.angle = 0
        return pp

    if p.x == 0:
        pp.distance = abs(p.y)
        if p.y > 0:
            pp.angle = 90
        else:
            pp.angle = 270
        return pp

    if p.y == 0:
        pp.distance = abs(p.x)
        if p.x > 0:
            pp.angle = 0
        else:
            pp.angle = 180
        return pp

    #  else...
    pp.distance = __xround(math.sqrt(p.x * p.x + p.y * p.y))
    pp.angle = math.degrees(math.atan(p.y / p.x))
    if p.x < 0:
        pp.angle = 180 + pp.angle

    while pp.angle < 0.0:
        pp.angle = 360 + pp.angle

    pp.angle = __xround(pp.angle)
    return pp
# END
```

For some reason - unbeknownst to me! - Python allows and returns the 'value -0.0' We convert this 'value' to zero via the absolute value function.

Class XYDegreeSpinner

File: Spinner.py

```
import PolarPoint
import XYUtils
from AngleMeasure import AngleMeasure
from Points import Points


class XYDegreeSpinner:
    def __init__(self, xyPoints: Points, rotateDegrees):
        self.rotateDegrees = rotateDegrees
        self.angleMeasure = AngleMeasure()
        self.angleMeasure.unit = AngleMeasure.DEGREES

        # convert Points to list of polar points
        self.pPointsList = []
        for xyPoint in xyPoints.asList():
            pPoint = XYUtils.xyToPolarPoint(xyPoint, self.angleMeasure)
            self.pPointsList.append(pPoint)

    def nextRotation(self):
        rotatedXYPoints = Points();

        for pPoint in self.pPointsList:
            # polar: to rotate we rotate the angle and leave distance unchanged
            a = PolarPoint.incrementAngle(pPoint.angle, self.rotateDegrees, self.angleMeasure)
            pPoint.angle = a
            xyPoint = XYUtils.polarPointToXY(pPoint, self.angleMeasure)
            rotatedXYPoints.append(xyPoint)

        return rotatedXYPoints

# eof
```

The END