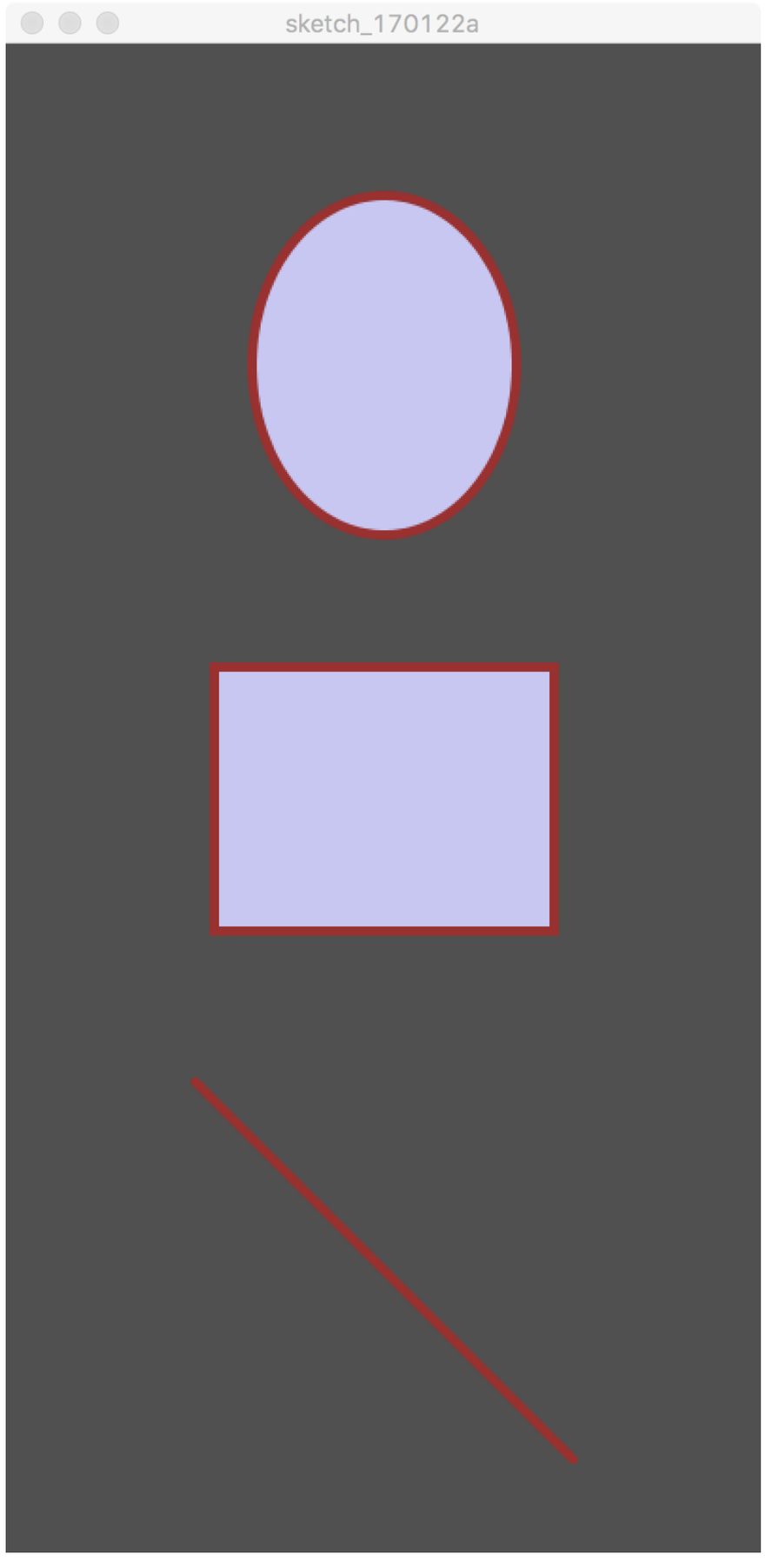
Module 04

Advanced Shapes

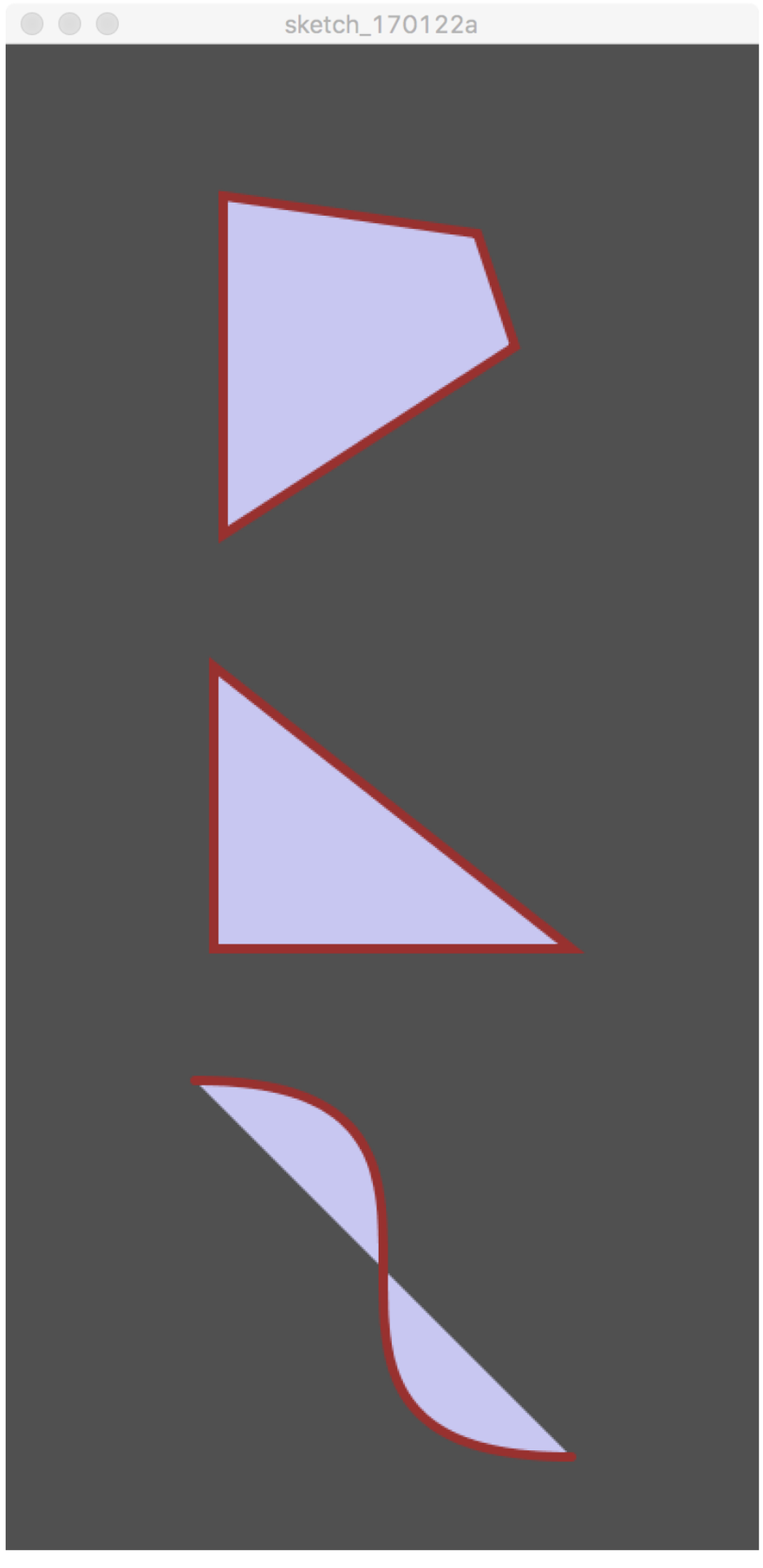
CS 106 Winter 2019

Hollow by Eva Hild

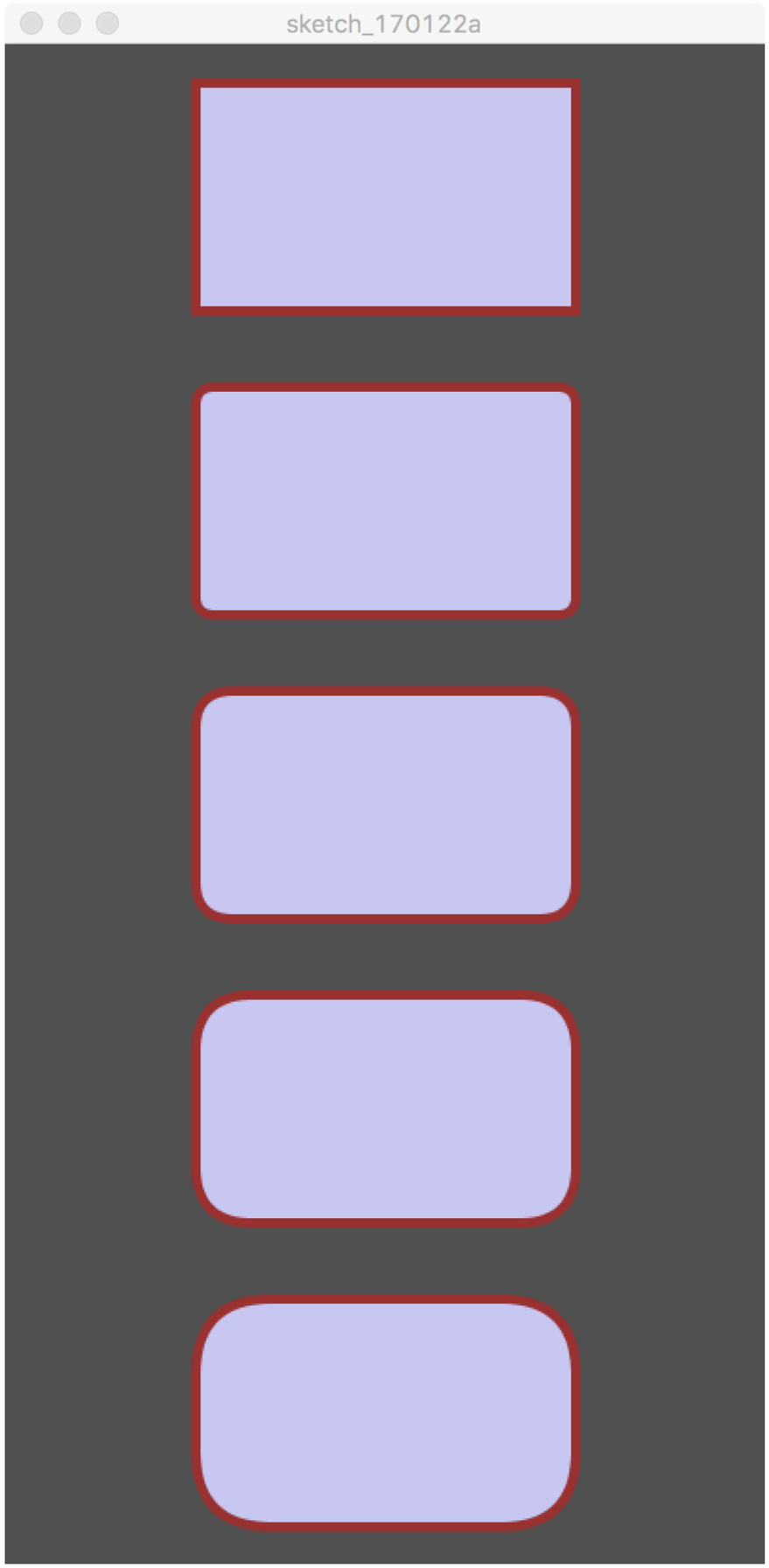
```
void setup()
{
  size( 400, 800 );
  background( 80 );
  rectMode( CENTER );

  fill( 200, 200, 240 );
  stroke( 150, 50, 50 );
  strokeWeight( 5 );

  ellipse( 200, 170, 140, 180 );
  rect( 200, 400, 180, 140 );
  line( 100, 550, 300, 750 );
}
```



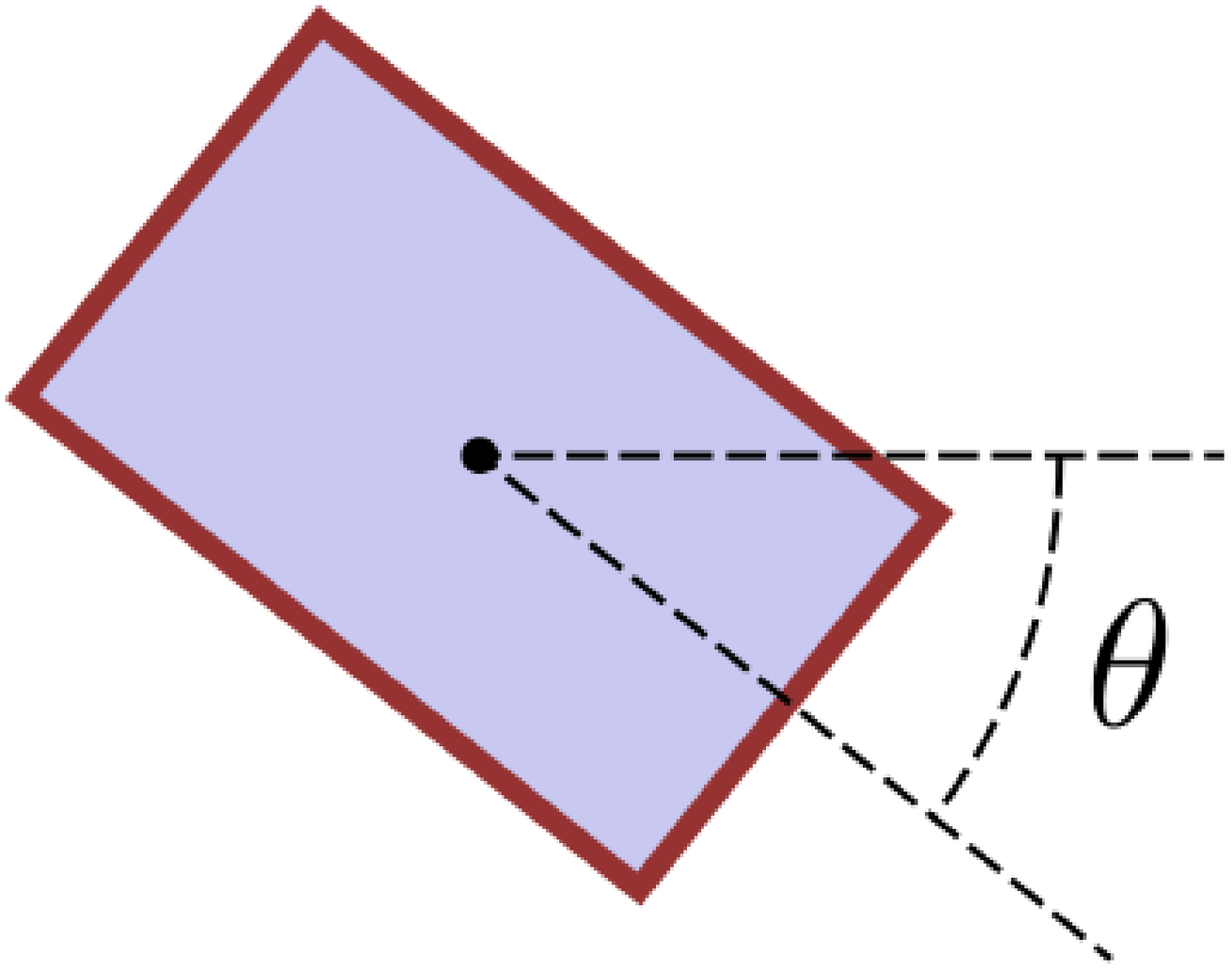
```
void setup()
{
  size( 400, 800 );
  background( 80 );
  rectMode( CENTER );

  fill( 200, 200, 240 );
  stroke( 150, 50, 50 );
  strokeWeight( 5 );

  quad( 115, 80, 115, 260,
    270, 160, 250, 100 );
  triangle( 110, 330, 110, 480,
    300, 480 );
  bezier( 100, 550, 300, 550,
    100, 750, 300, 750 );
}
```

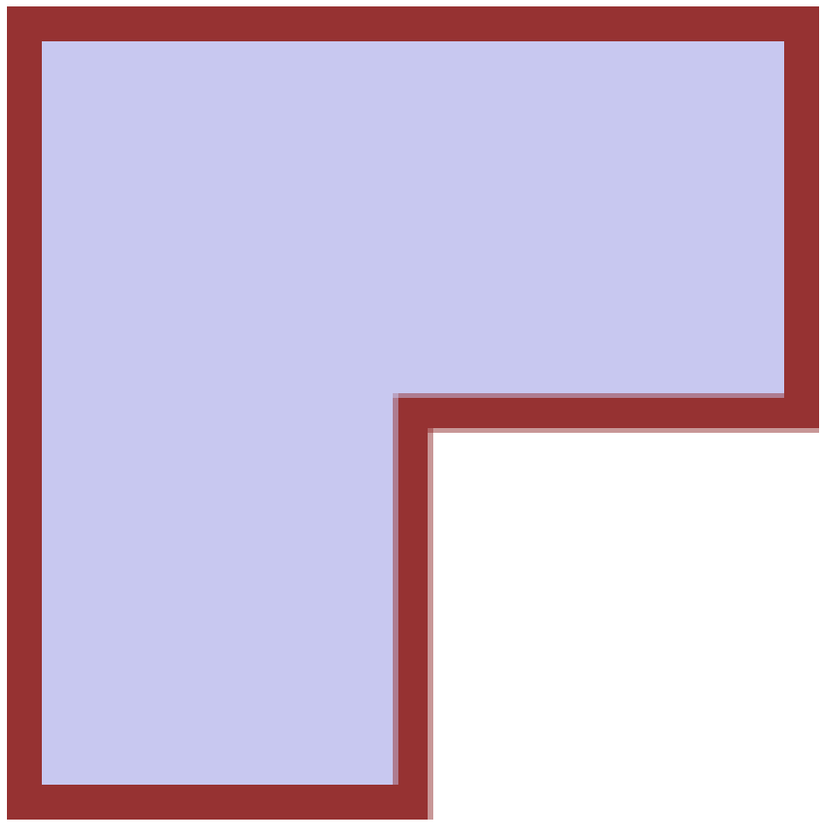


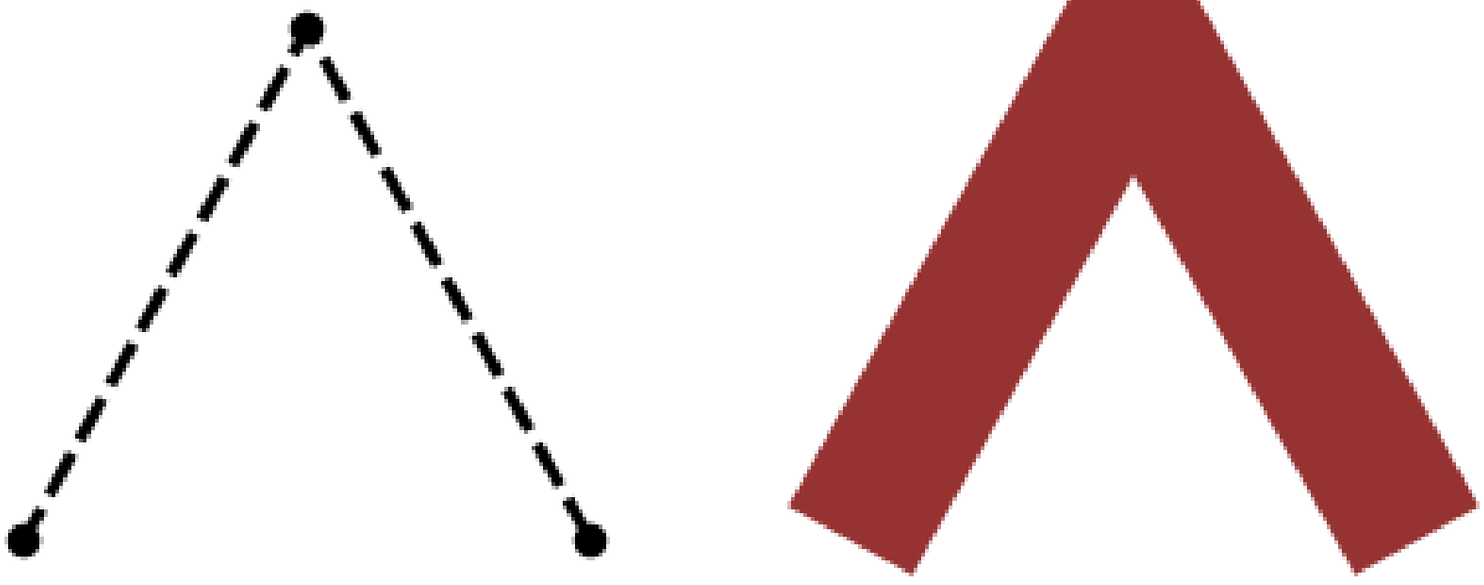
```
void setup()
{
  size( 400, 800 );
  background( 80 );

  fill( 200, 200, 240 );
  stroke( 150, 50, 50 );
  strokeWeight( 5 );

  for( int idx = 0; idx < 5; ++idx ) {
    rect( 100, idx * 160 + 20,
      200, 120, idx * 10 );
  }
}
```

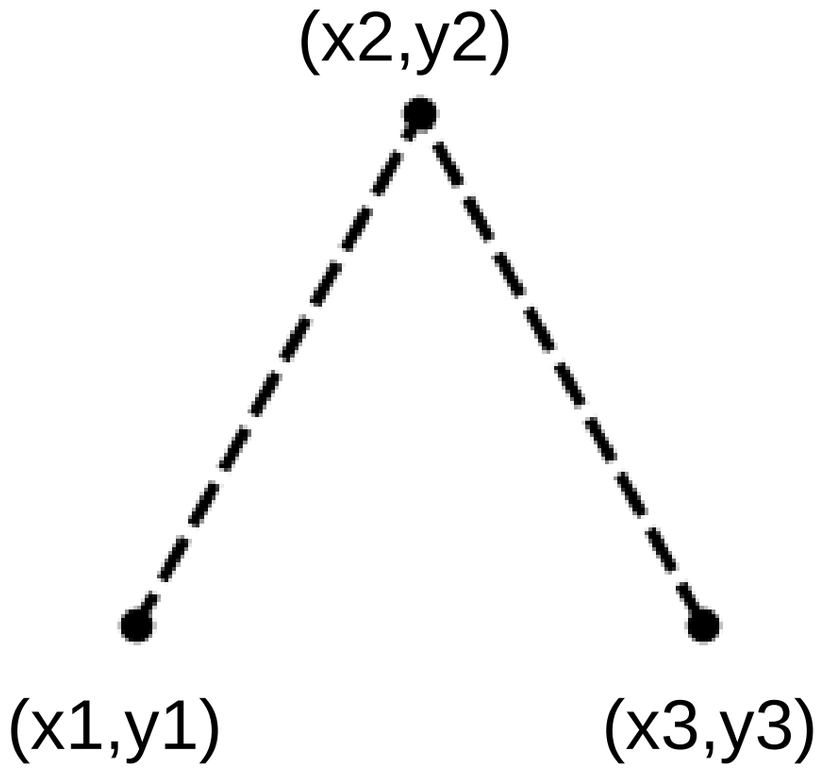


Limitations

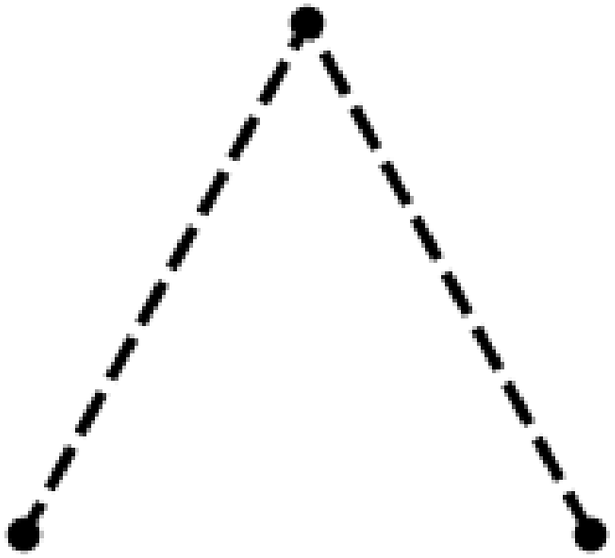








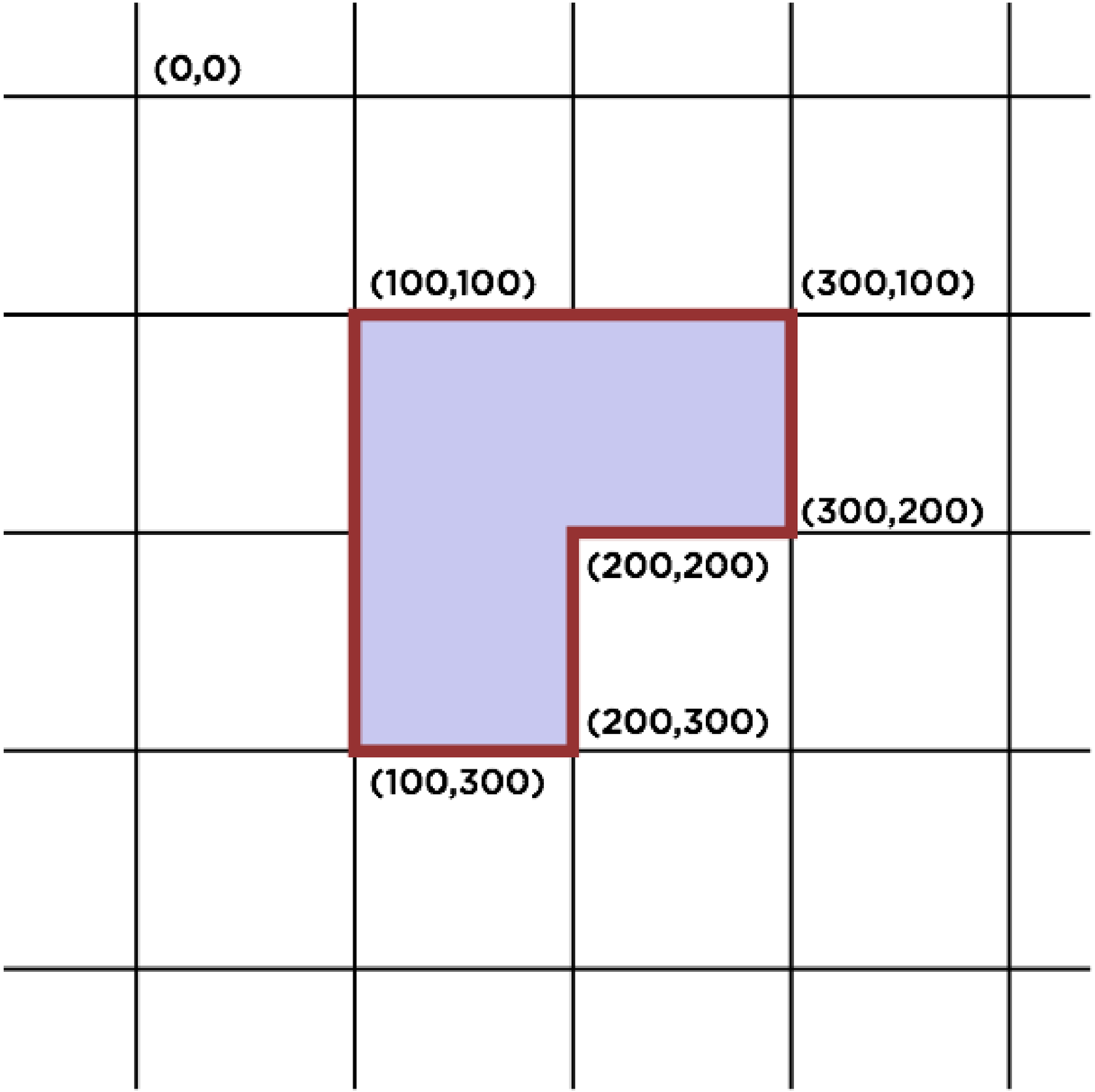
```
line( x1, y1, x2, y2 );
line( x2, y2, x3, y3 );
```



strokeCap(SQUARE)

strokeCap(PROJECT)

strokeCap(ROUND)

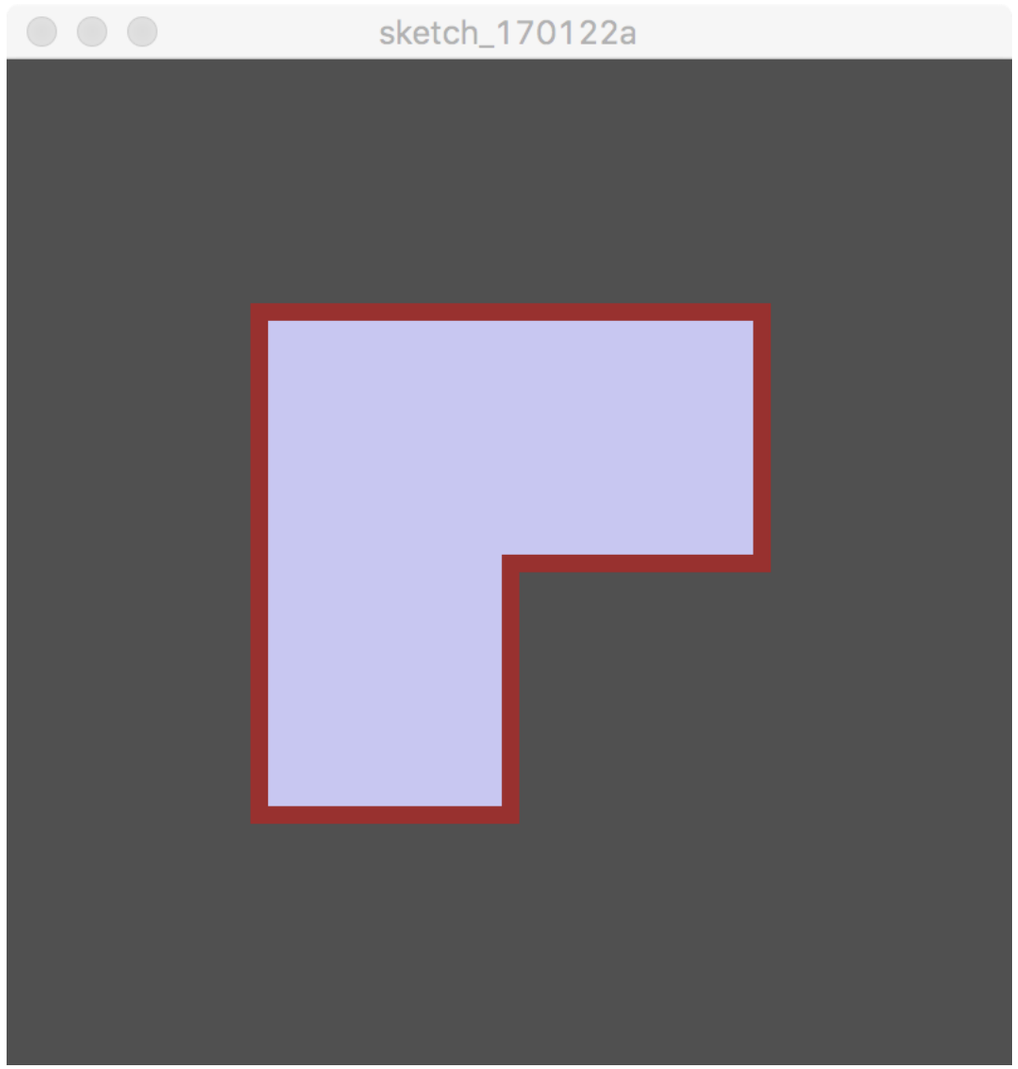

(0,0)
(100,100)
(300,100)
(300,200)
(200,200)
(200,300)
(100,300)

beginShape() “I will now draw a shape.”

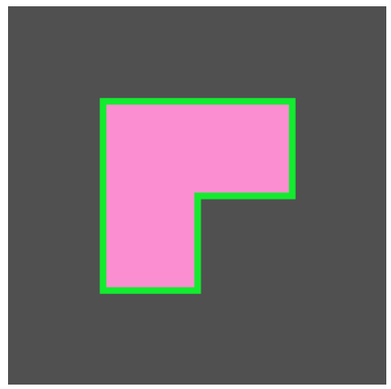
vertex() “Here’s one corner of that shape I’m drawing.”

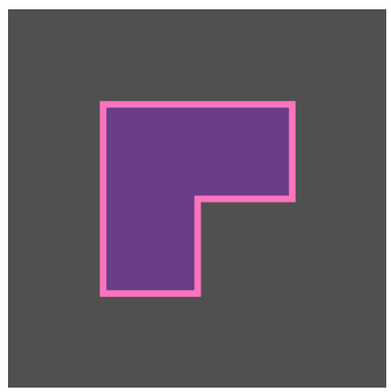
endShape() “Now I am finished drawing my shape.”

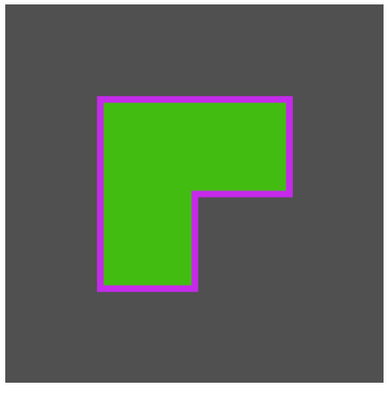
```
beginShape();
  vertex( 100, 100 );
  vertex( 100, 300 );
  vertex( 200, 300 );
  vertex( 200, 200 );
  vertex( 300, 200 );
  vertex( 300, 100 );
endShape( CLOSE );
```

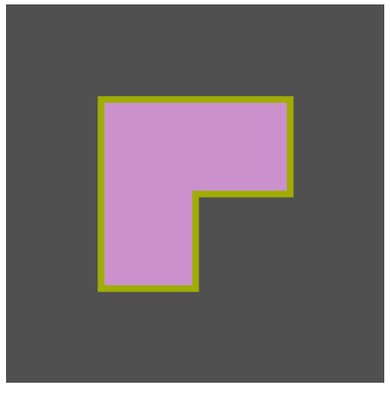


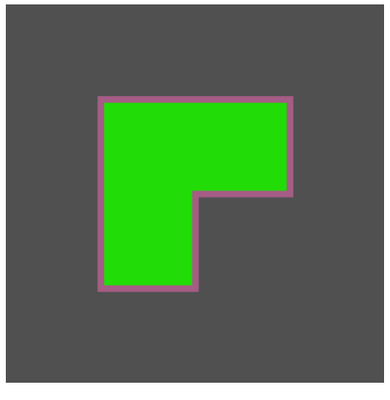
```
for ( int idx = 0; idx < 8; ++idx ) {
  fill( random(255), random(255), random(255) );
  stroke( random(255), random(255), random(255) );
  background( 80 );

  beginShape();
    vertex( 100, 100 );
    vertex( 100, 300 );
    vertex( 200, 300 );
    vertex( 200, 200 );
    vertex( 300, 200 );
    vertex( 300, 100 );
  endShape( CLOSE );

  save( "output-" + idx + ".png" );
}
```

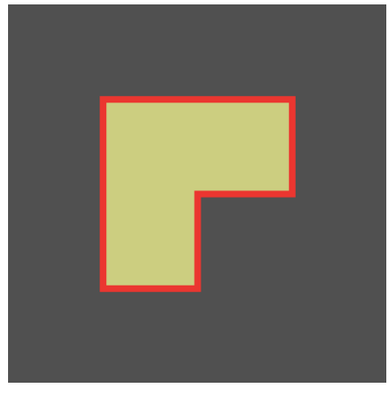








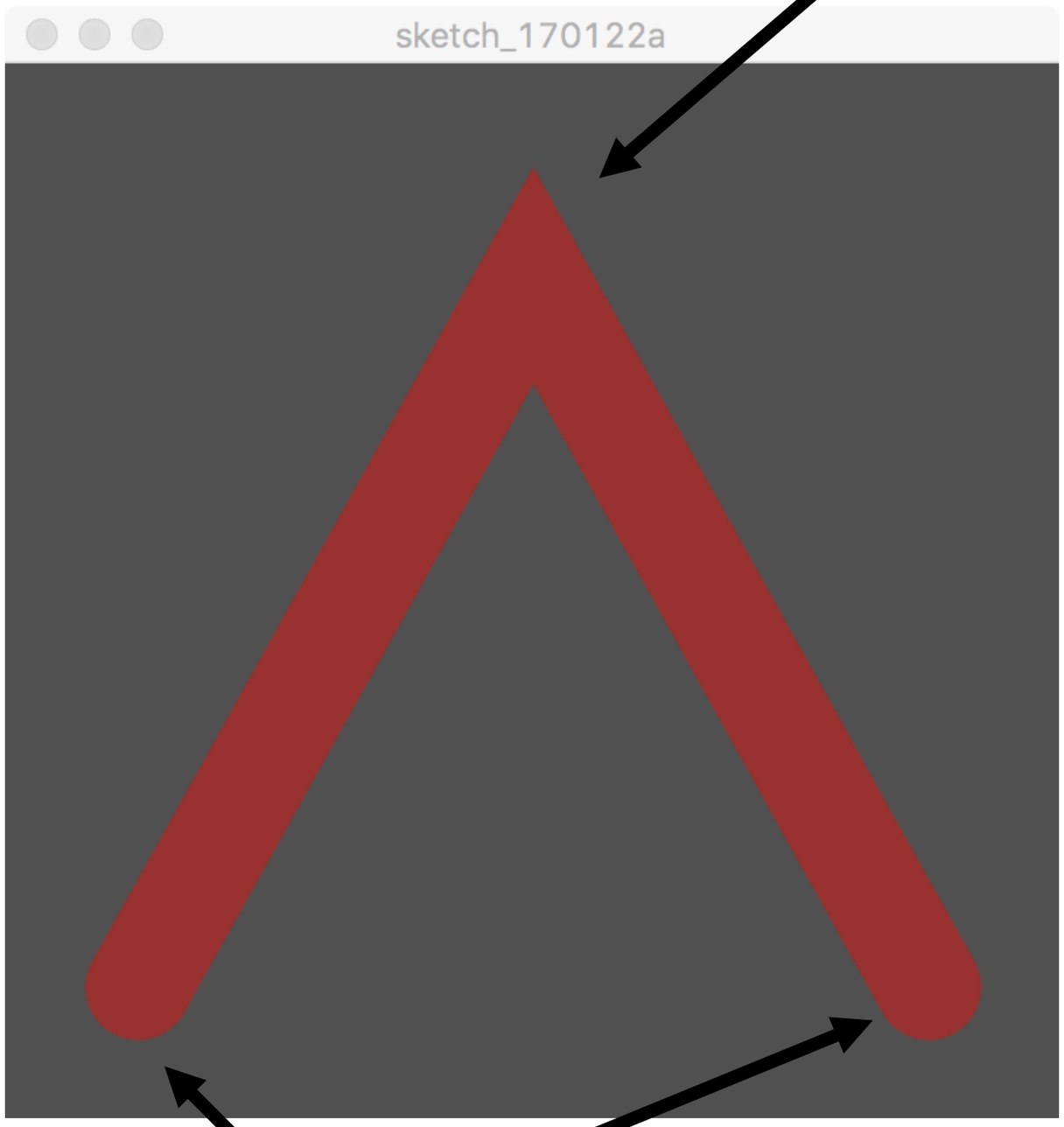




```
float[] coords = {
  100, 100,  100, 300,  200, 300,
  200, 200,  300, 200,  300, 100
};

beginShape();
  for ( int idx = 0; idx < coords.length; idx += 2 ) {
    vertex( coords[idx], coords[idx+1] );
  }
endShape( CLOSE );
```

```
beginShape();
  vertex( 50, 350 );
  vertex( 200, 80 );
  vertex( 350, 350 );
endShape();
```

strokeJoin()

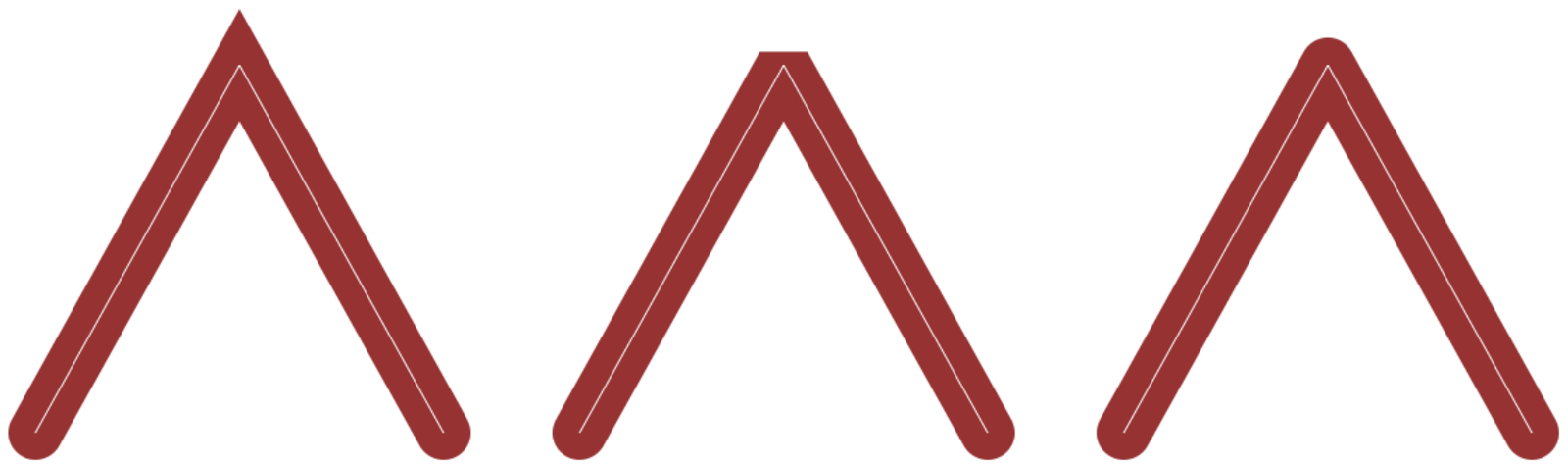


strokeCap()

strokeCap(SQUARE)

strokeCap(PROJECT)

strokeCap(ROUND)

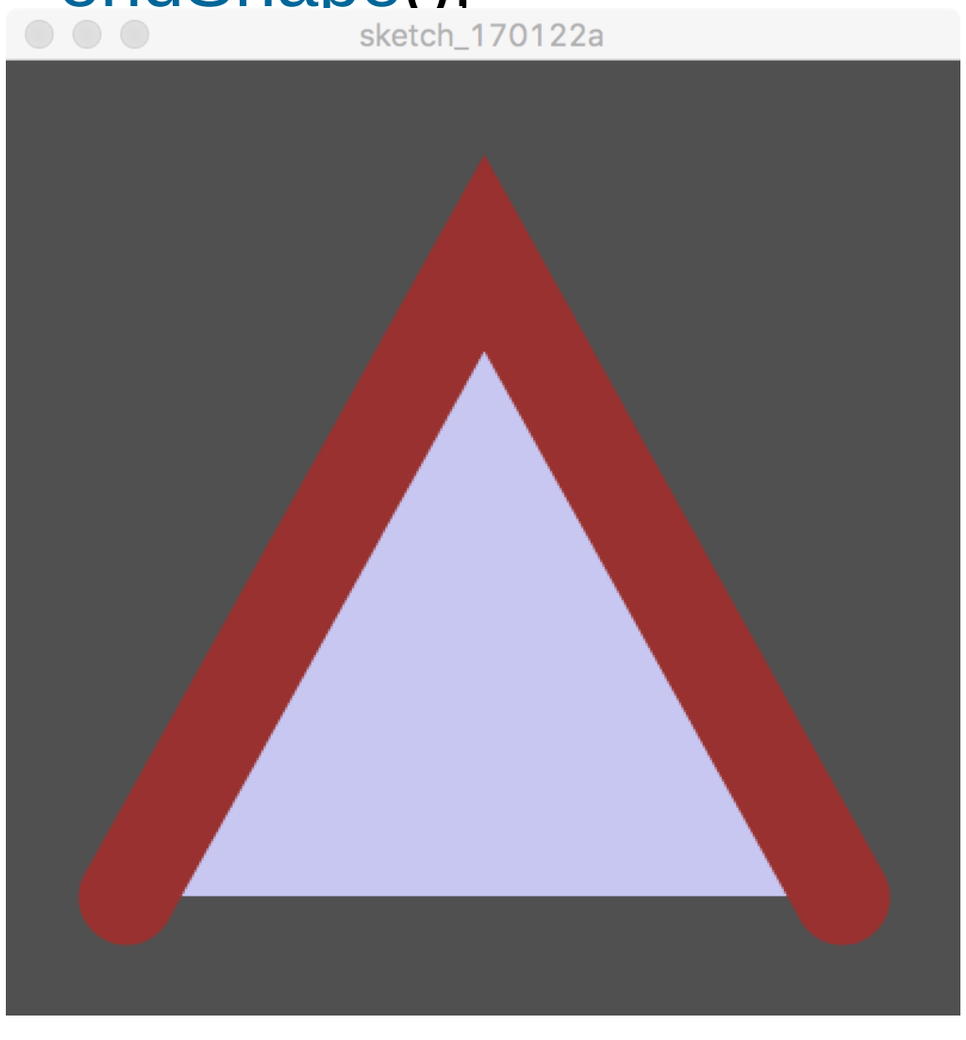


strokeJoin(MITER)

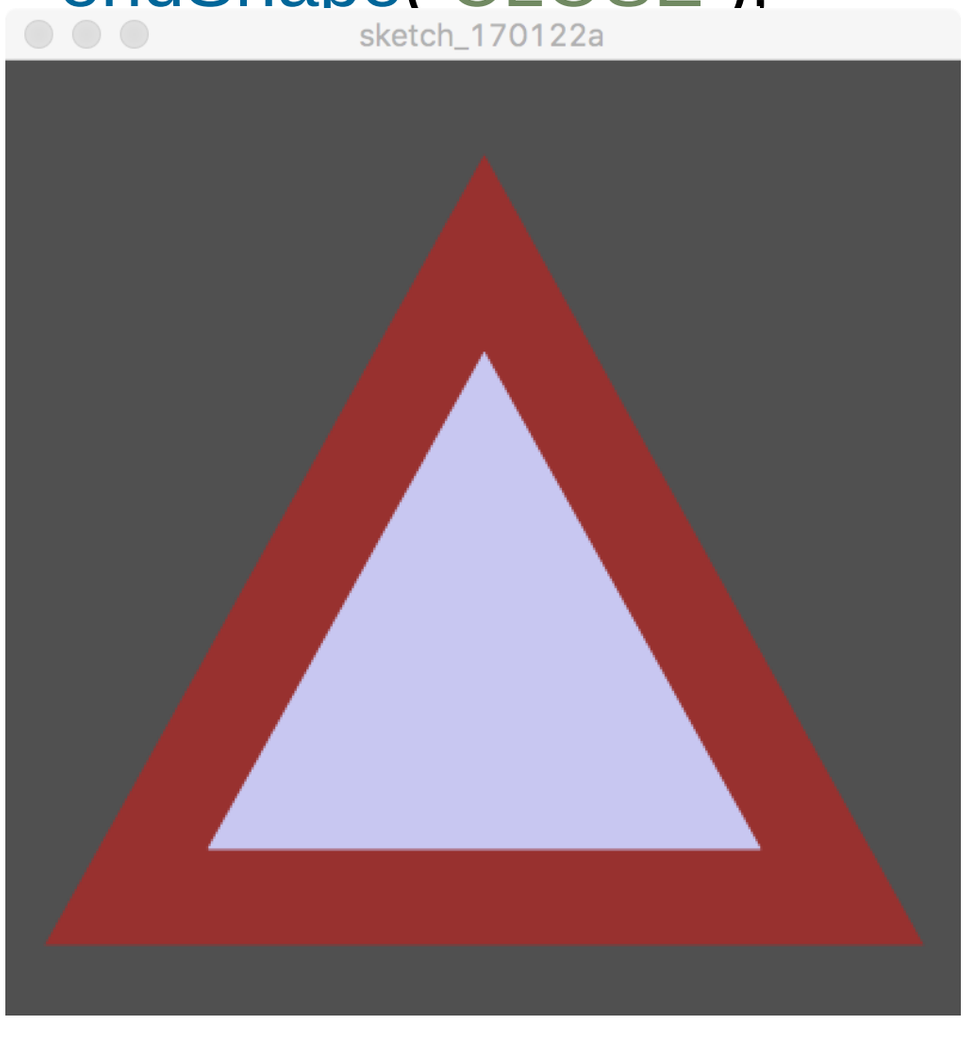
strokeJoin(ROUND)

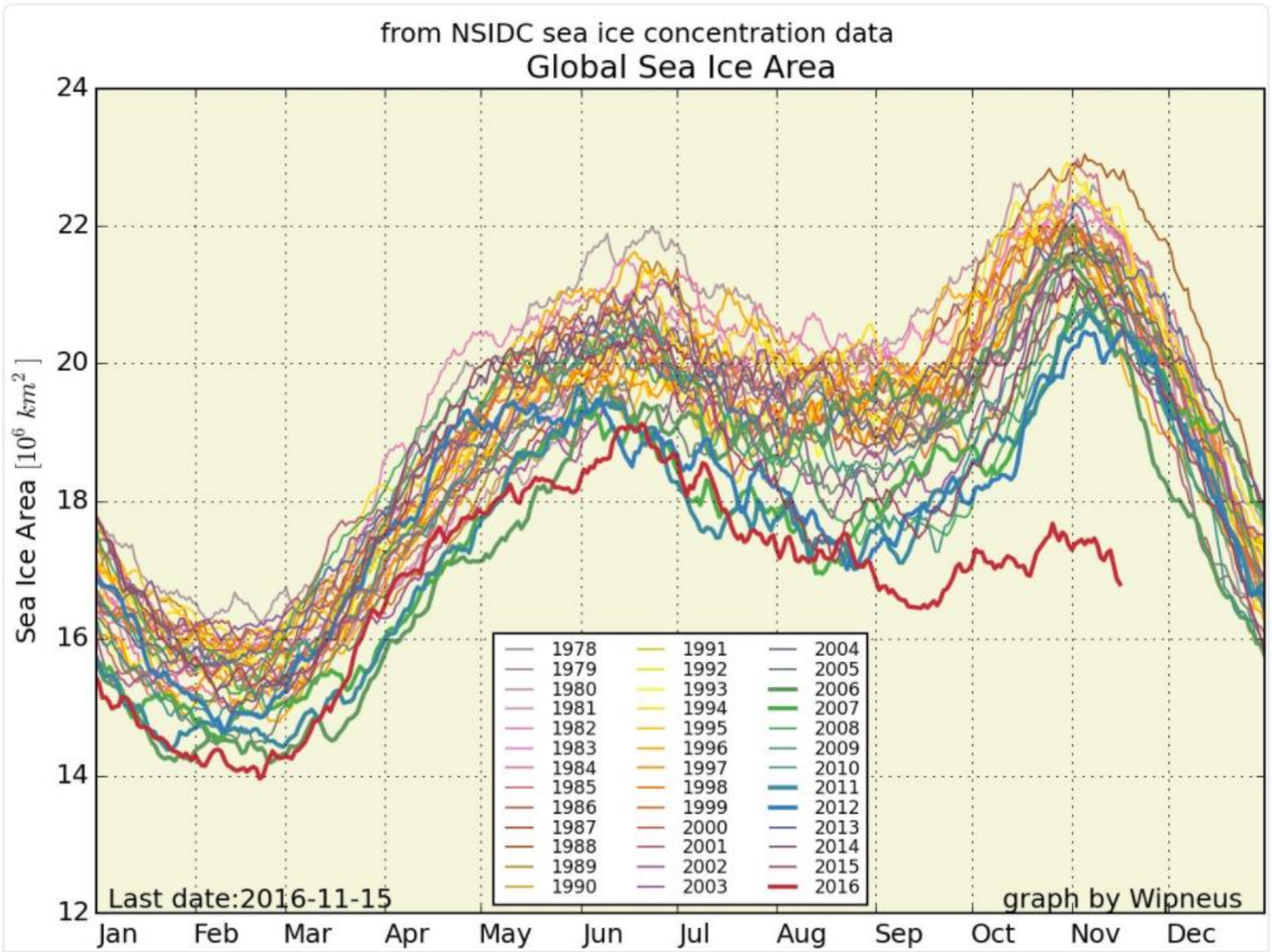
strokeJoin(BEVEL)

```
fill( 200, 200, 240 );
stroke( 150, 50, 50 );
strokeWeight( 40 );

beginShape();
vertex( 50, 350 );
vertex( 200, 80 );
vertex( 350, 350 );
endShape();
```




```
fill( 200, 200, 240 );
stroke( 150, 50, 50 );
strokeWeight( 40 );

beginShape();
vertex( 50, 350 );
vertex( 200, 80 );
vertex( 350, 350 );
endShape( CLOSE );
```





Zack Labe
@ZLabe

Follow

This is not normal. Global #seaice area...

(via sites.google.com/site/arctische…)

10:38 AM - 16 Nov 2016 · Irvine, CA

904 438

Sea Ice Index

NOTICE: Sea Ice Index now Version 2.1. Read more...

Sea Ice Index Data and Image Archive

Here you can access the archived monthly and daily Sea Ice Index images and data as well as the input data that the Sea Ice Index is derived from.

All archived data reside in the following FTP location: ftp://sidads.colorado.edu/DATASETS/NOAA/G02135/

The Sea Ice Index archive contains the following:

- Monthly images of extent, concentration, anomalies, and trends from November 1978 through last month.
  Note: Daily images are not archived, however, the most recent daily image can be obtained from the Sea Ice Index Web page.
- Monthly sea ice extent and area data files.
- Monthly GIS compatible shapefiles of ice extent.
- Daily sea ice extent data files and daily climatology data files with data from November 1978 through yesterday.

Images | Data | Input Data

Monthly Extent and Concentration Images

Only monthly image files are available in the archive. Daily images are not archived.

Access files:

1. Go to the FTP directory.

https://nsidc.org/data/seaice_index/archives.html

1979 15.6 16.38 16.52 15.56 14.09 12.65 10.52 8.18 7.22 9.43 11.19 13.58
1980 15.05 16.07 16.21 15.55 14.08 12.37 10.43 8.08 7.86 9.49 11.72 13.75
1981 15.11 15.75 15.7 15.19 13.92 12.62 10.67 7.89 7.25 9.21 11.21 13.78
1982 15.34 16.14 16.23 15.62 14.18 12.75 10.81 8.3 7.45 10.02 11.96 13.87
1983 15.16 16.1 16.19 15.36 13.57 12.39 10.95 8.39 7.54 9.69 11.68 13.49
1984 14.69 15.42 15.72 15.23 13.7 12.25 10.22 7.94 7.11 8.87 11.32 13.21
1985 14.96 15.77 16.14 15.41 14.26 12.45 10.16 7.5 6.94 8.92 11.42 13.25
1986 15.12 15.99 16.16 15.23 13.55 12.14 10.5 8.04 7.55 9.92 11.81 13.46
1987 15.28 16.21 16.02 15.38 13.85 12.61 10.81 7.72 7.51 9.33 11.59 13.6
1988 15.25 15.73 16.23 15.32 13.75 12.11 10.12 7.99 7.53 9.51 11.73 13.87
1989 15.23 15.67 15.62 14.53 13.05 12.37 10.47 8.0 7.08 9.55 11.57 13.57
1990 15.07 15.69 15.99 14.79 13.36 11.77 9.68 6.87 6.27 9.4 11.37 13.36
1991 14.58 15.37 15.6 15.03 13.58 12.29 9.76 7.46 6.59 9.2 11.16 13.21
1992 14.86 15.62 15.62 14.8 13.31 12.2 10.7 7.96 7.59 9.64 11.91 13.53
1993 15.2 15.85 15.98 15.3 13.6 12.07 9.74 7.39 6.54 9.25 11.78 13.58
1994 14.94 15.72 15.71 15.07 13.82 12.16 10.3 7.7 7.24 9.53 11.35 13.61
1995 14.74 15.37 15.44 14.71 13.22 11.66 9.23 6.75 6.18 8.98 11.05 13.09
1996 14.35 15.32 15.25 14.35 13.14 12.17 10.46 8.29 7.91 9.43 10.61 13.21
1997 14.62 15.63 15.69 14.69 13.39 11.99 9.66 7.35 6.78 8.8 10.94 13.35
1998 14.94 15.9 15.78 14.99 13.86 11.95 9.67 7.56 6.62 8.89 10.79 13.34
1999 14.57 15.48 15.52 15.24 13.92 12.19 9.65 7.44 6.29 9.14 11.03 12.95
2000 14.54 15.3 15.38 14.75 13.26 11.8 9.81 7.27 6.36 8.95 10.58 12.89
2001 14.42 15.42 15.72 14.97 13.78 11.75 9.28 7.53 6.78 8.6 10.94 12.92
2002 14.57 15.47 15.54 14.46 13.18 11.77 9.58 6.57 5.98 8.83 10.8 12.89

Demo Code

loadString()

SplitTokens()

Go through in detail Demo Code:

SeaIceJanuary

```
    float[] samps = float( loadStrings( "january.txt" ) );
```

SeaIceMonthly

```
    float[] vals = subset( float( splitTokens( line ) ), 1, 12 );
```

Representing points

It’s easy to pass a point to a function: just pass *x* and *y* separately.

```
vertex( 37, 192 );
```

Can we write a function that *returns* a point?

```
[redacted] makeAPoint() { ... }
```

```
[illegible] sampleUnitSquare()
{
   float x = random( 1 );
   float y = random( 1 );

   return [illegible] [illegible] x [illegible] y [illegible];
}
```

```
float[] sampleUnitSquare()
{
  float x = random( 1 );
  float y = random( 1 );
  float[] result = { x, y };
  return result;
}
```

```
class Point
{
  float x;
  float y;

  Point( float xIn, float yIn )
  {
    x = xIn;
    y = yIn;
  }
}
```

```
class Point{
  float x;
  float y;

  Point( float xIn, float yIn )
  {
    x = xIn;
    y = yIn;
  }
}

Point sampleUnitSquare() {
  float x = random( 1 );
  float y = random( 1 );

  return new Point( x, y );
}
```

```
void setup()
{
  for( int idx = 0; idx < 100; ++idx ) {
    Point pt = sampleUnitSquare();
    ellipse( pt.x * 100, pt.y * 100, 10, 10 );
  }
}
```

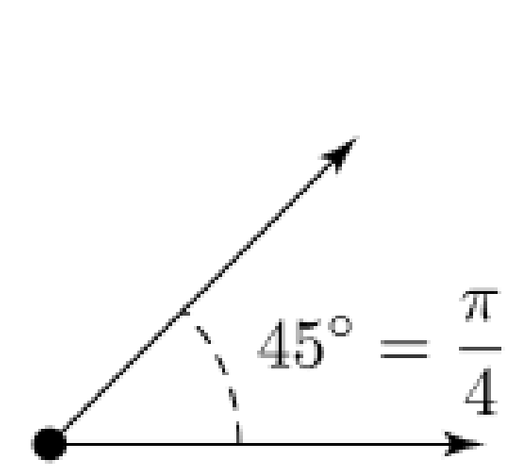
Processing offers a built-in class PVector for representing points in 2D (and 3D!).

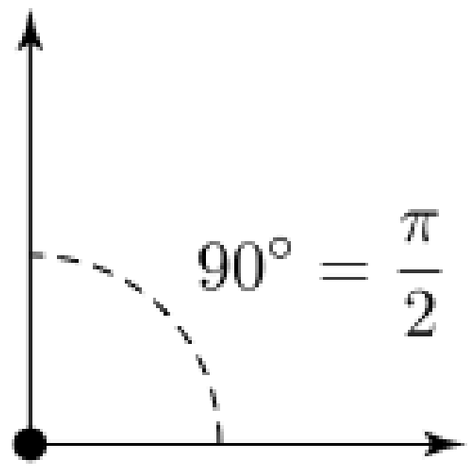
```
PVector sampleUnitSquare()
{
  float x = random( 1 );
  float y = random( 1 );

  return new PVector( x, y );
}

void setup()
{
  for( int idx = 0; idx < 100; ++idx ) {
    PVector pt = sampleUnitSquare();
    ellipse( pt.x * 100, pt.y * 100, 10, 10 );
  }
}
```

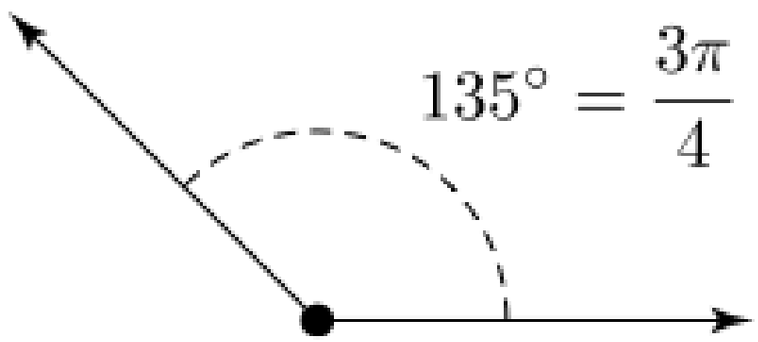
```
PVector[] pts = {
  new PVector( 100, 100 ),
  new PVector( 100, 300 ),
  new PVector( 200, 300 ),
  new PVector( 200, 200 ),
  new PVector( 300, 200 ),
  new PVector( 300, 100 )
};

beginShape();
for ( int idx = 0; idx < pts.length; ++idx ) {
  vertex( pts[idx].x, pts[idx].y );
}
endShape( CLOSE );
```

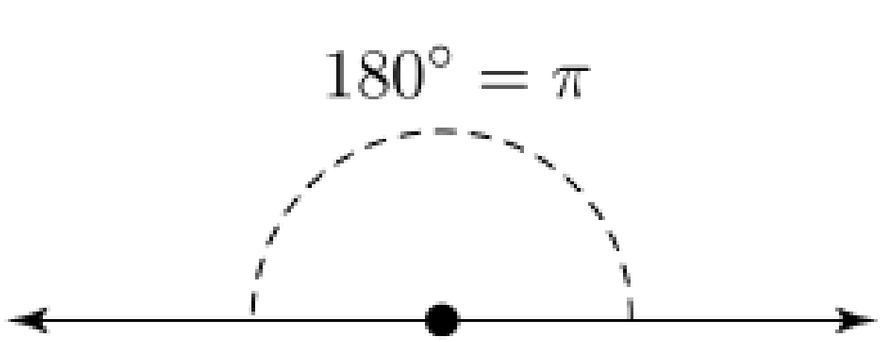
Angles

An *angle* is the measurement of the opening between two lines. We learn to measure angles in degrees, but programming languages usually expect them in *radians*.

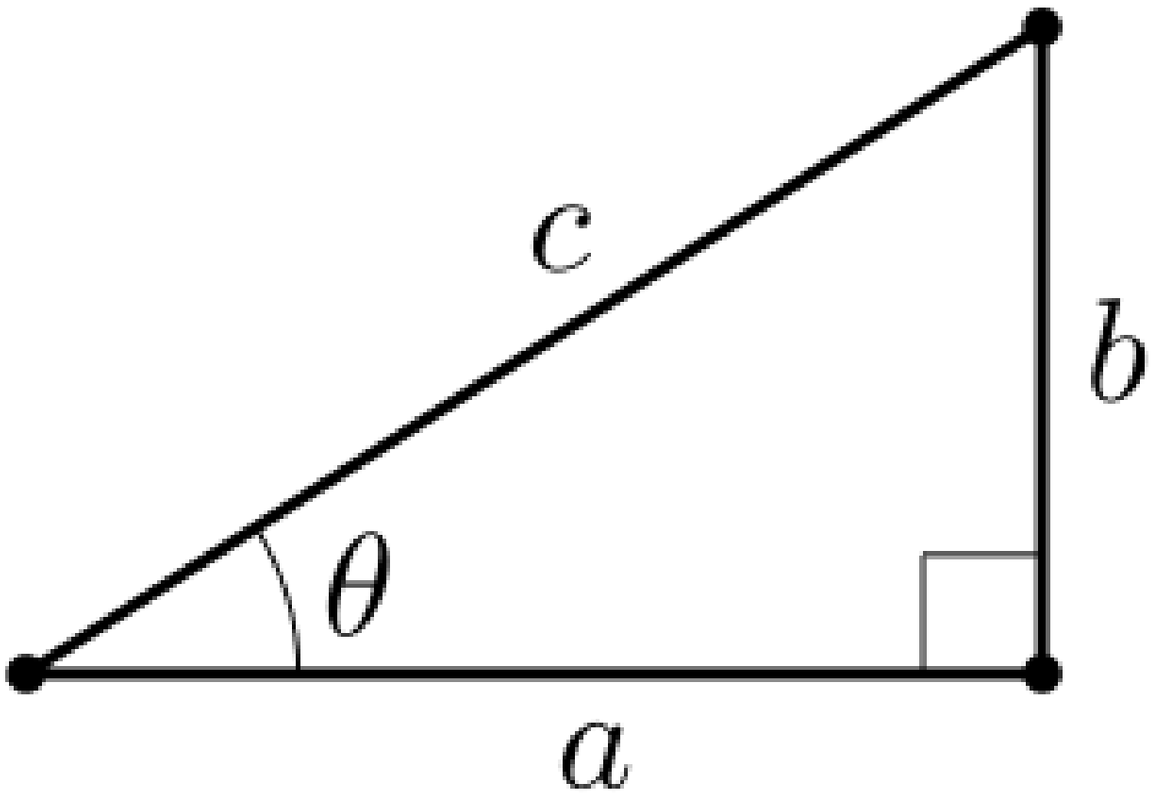




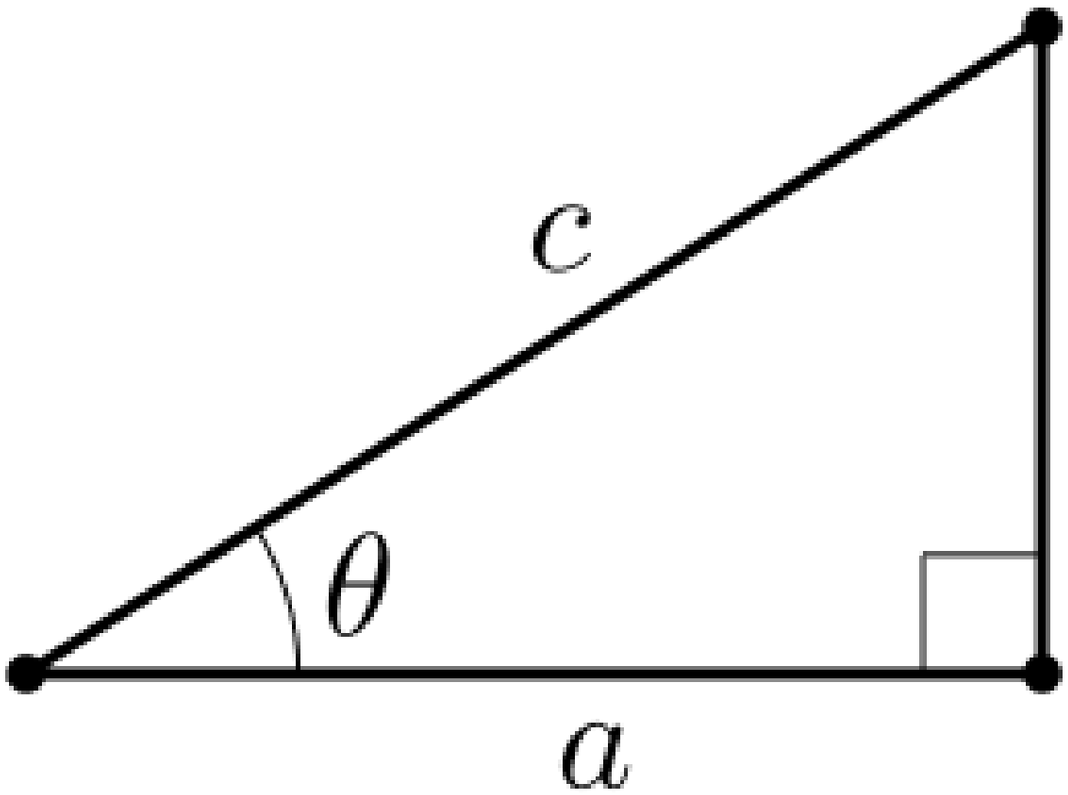




Trigonometry



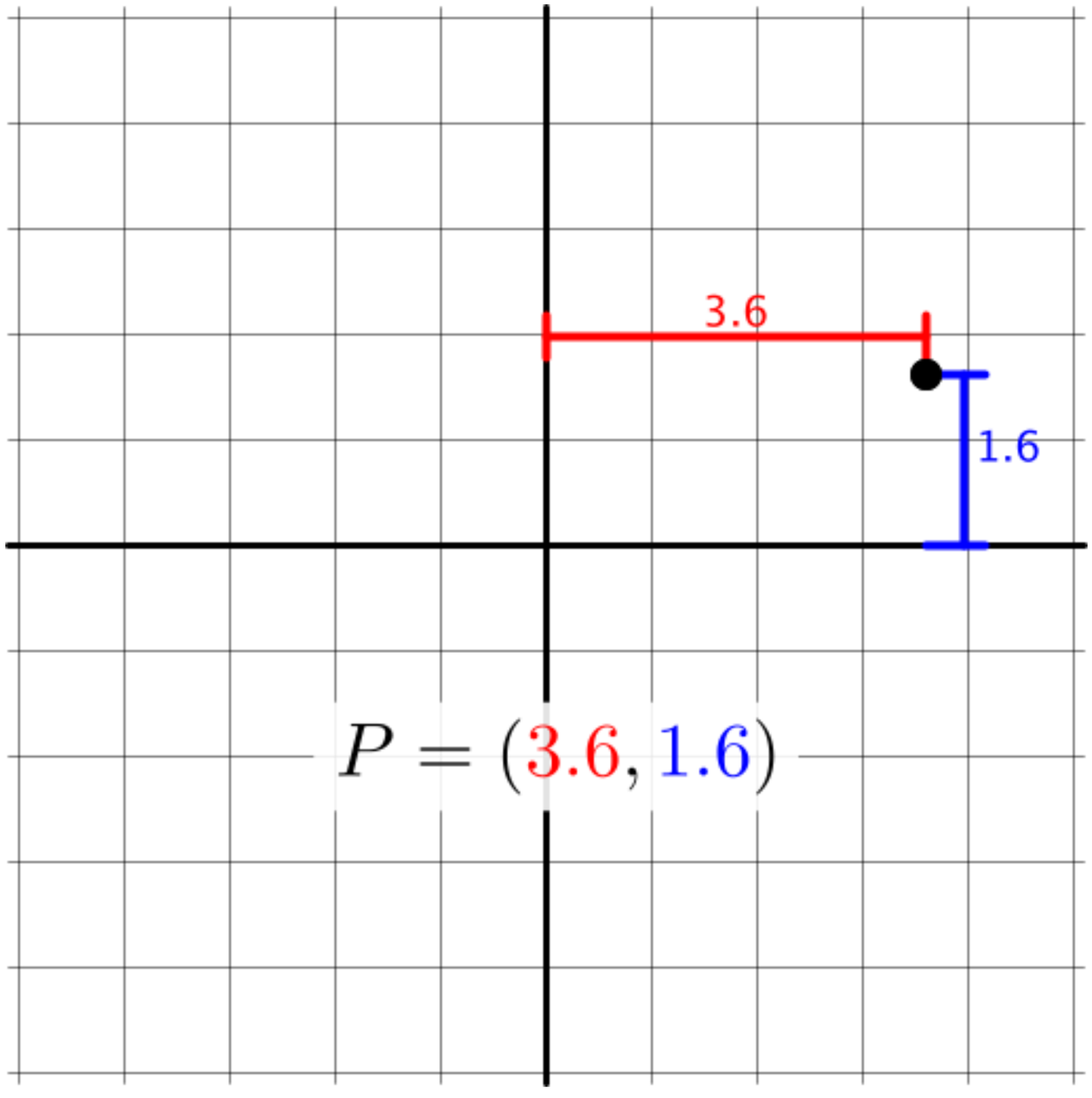
Trigonometry



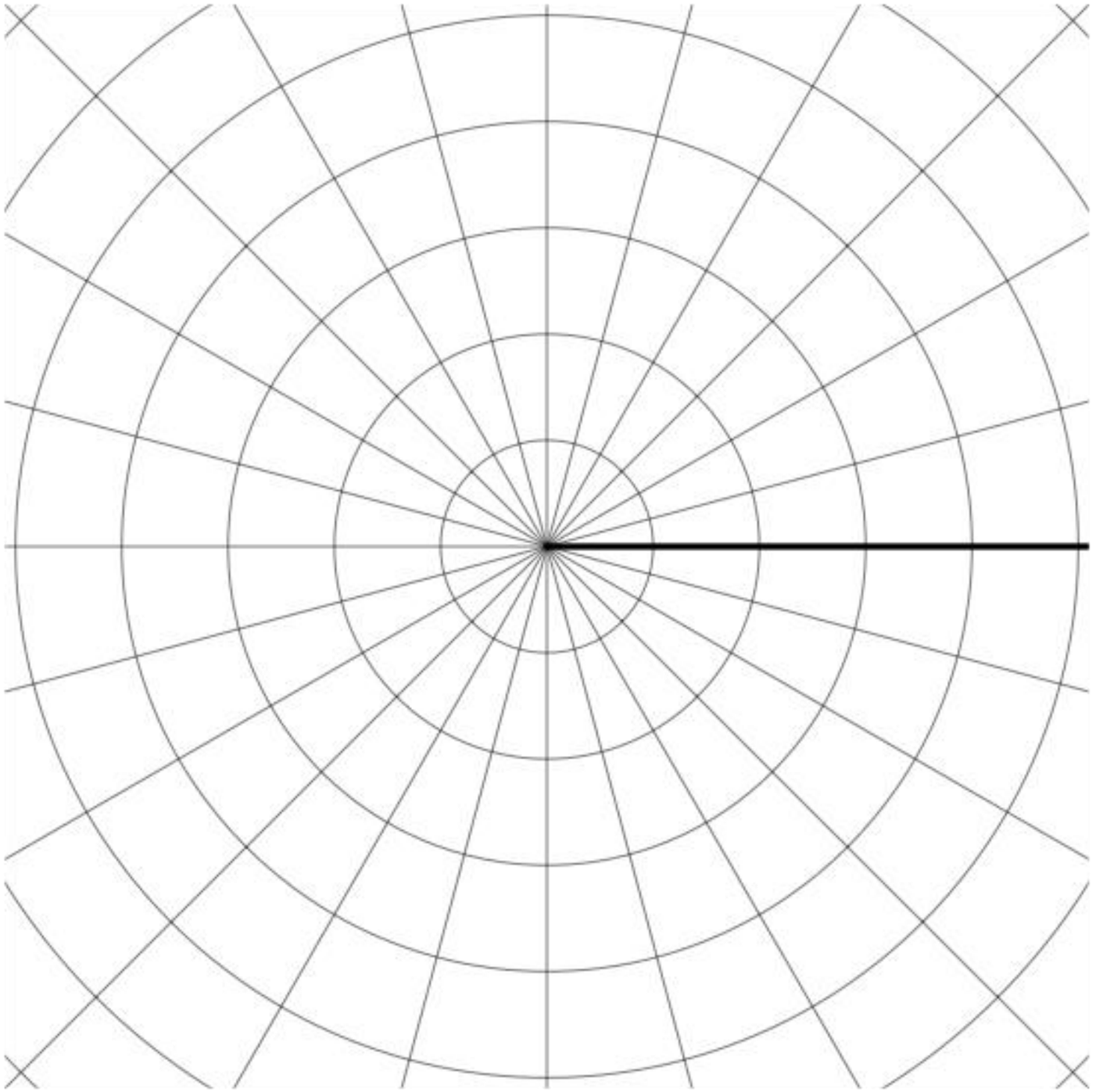
$$\sin\theta = \frac{b}{c}$$

$$\cos\theta = \frac{a}{c}$$

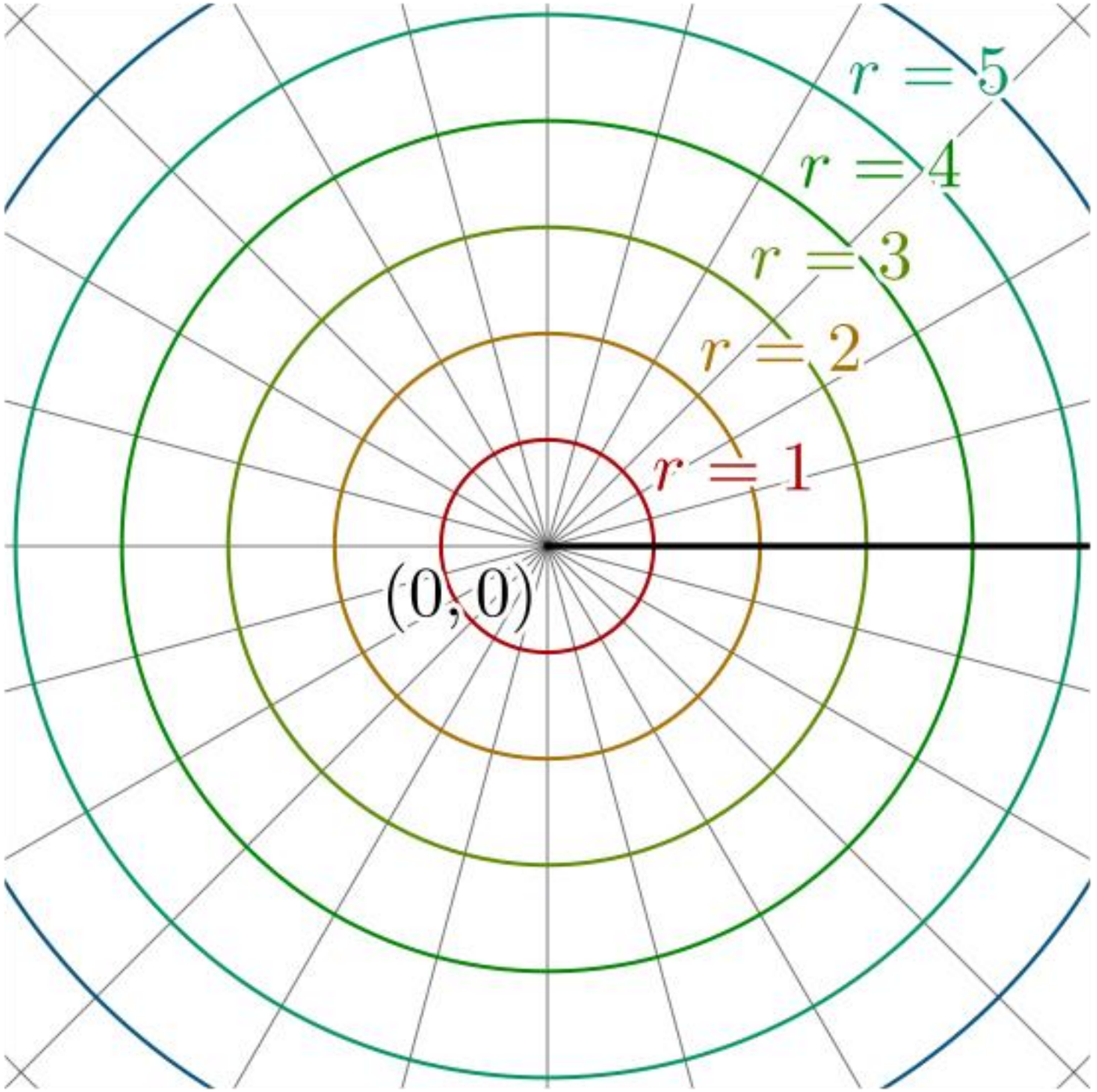
Cartesian (i.e., regular) coordinates



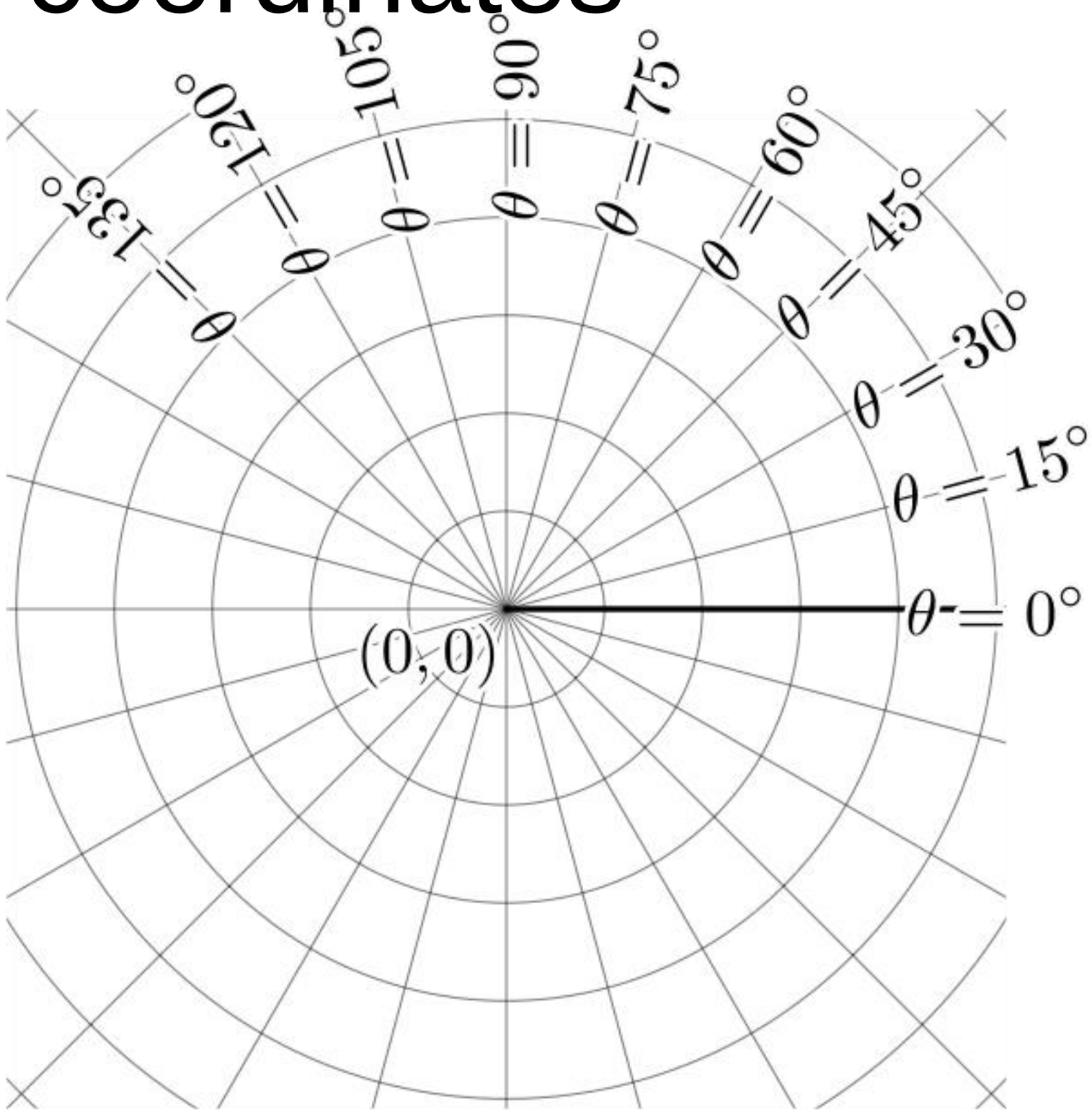
Polar coordinates



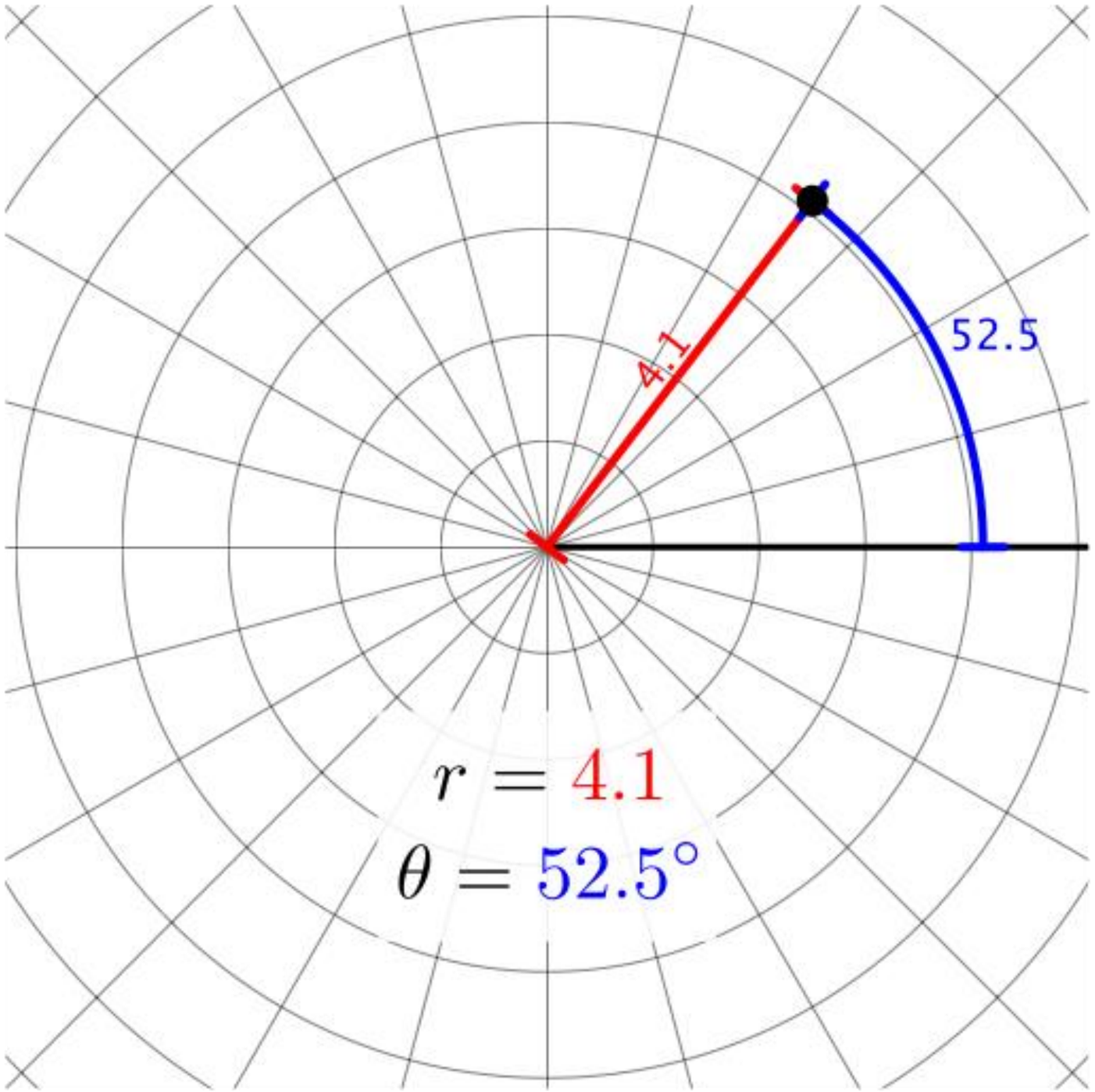
Polar coordinates

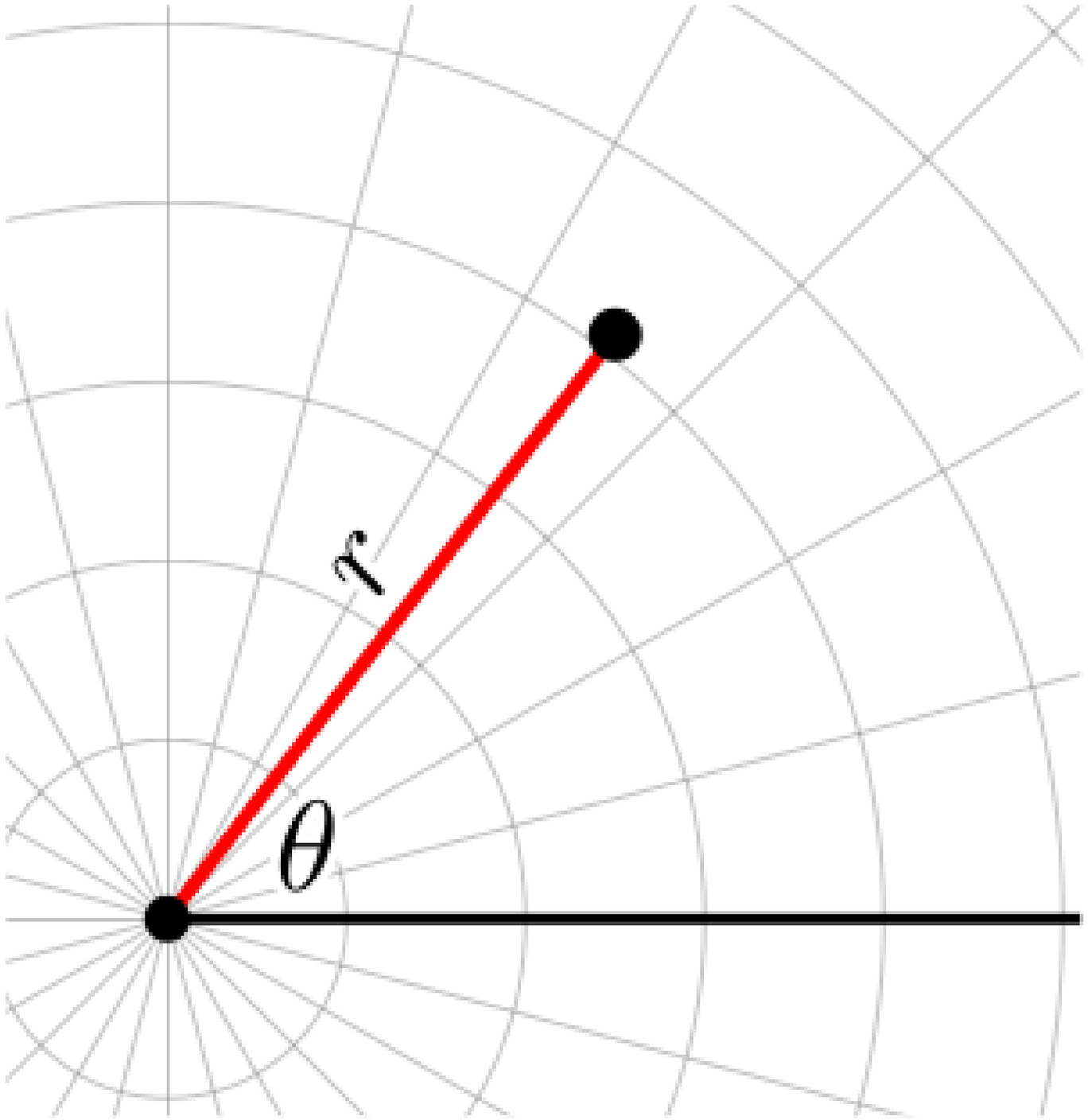


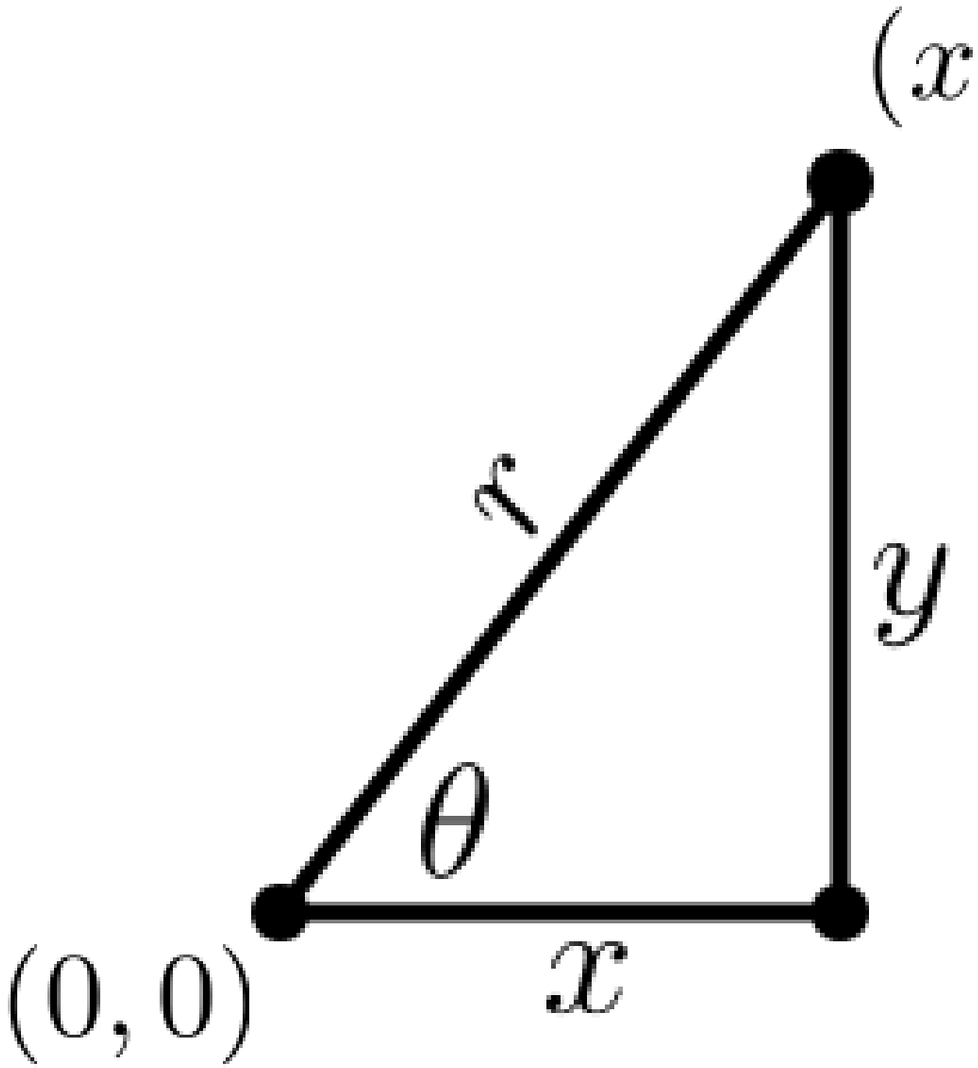
Polar coordinates



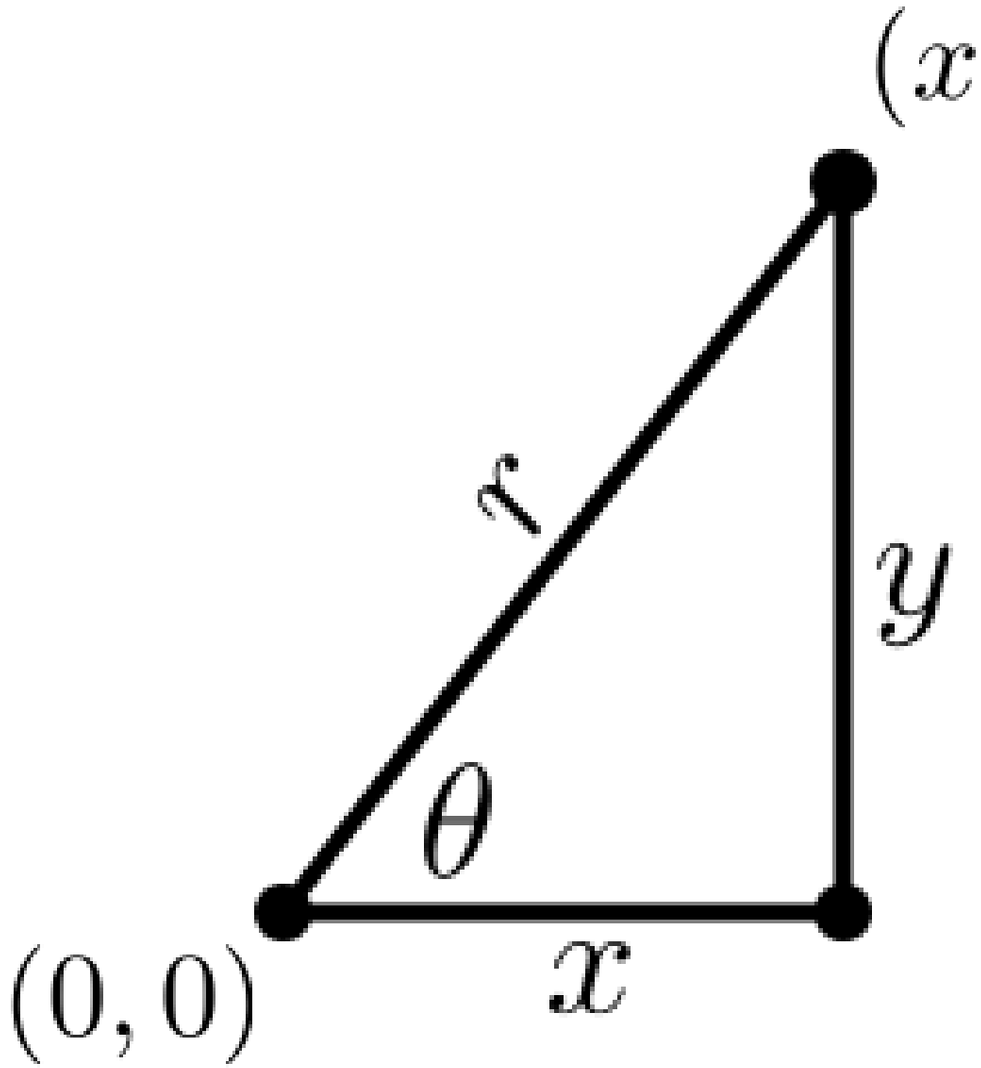
Polar coordinates



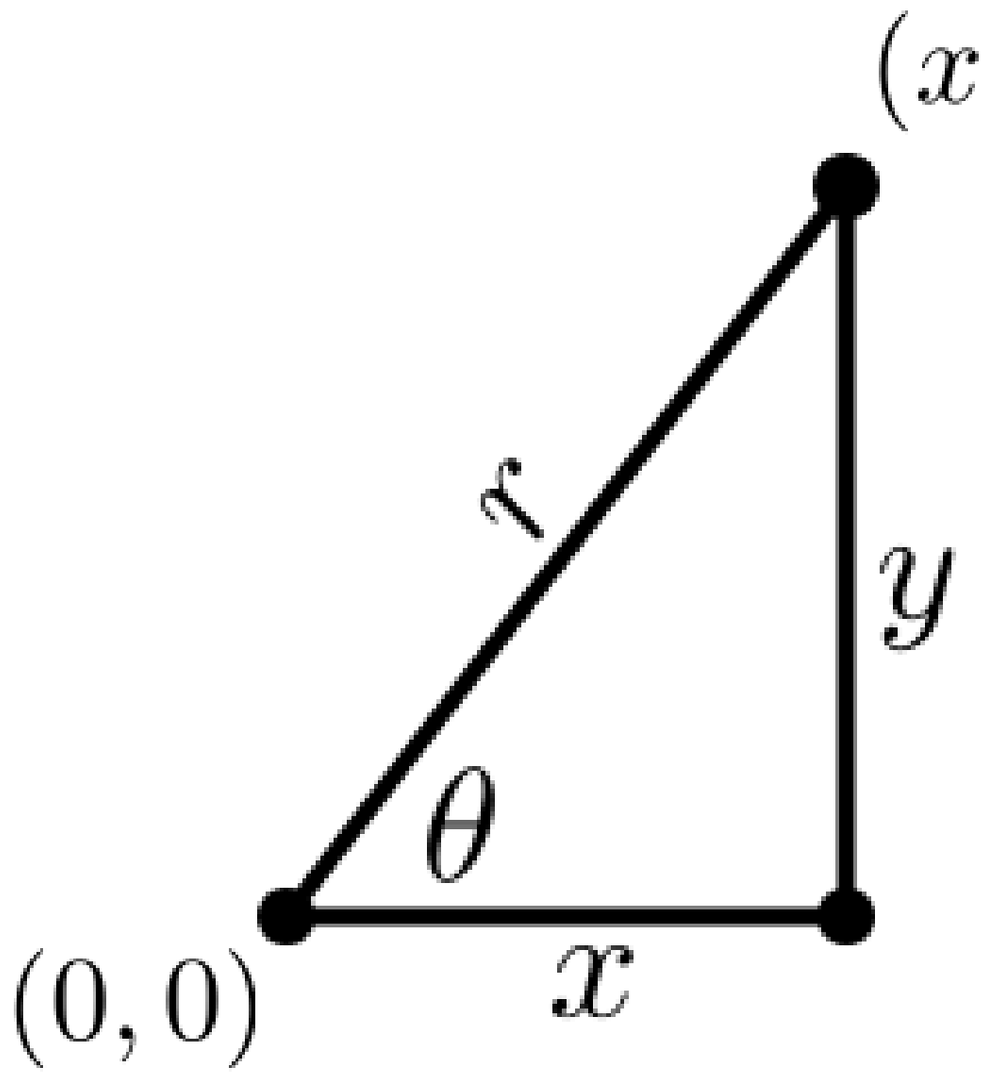

r
θ



$$\cos\theta = \frac{x}{r}$$



$$x = r\cos\theta$$



$$x = r\cos\theta$$
$$y = r\sin\theta$$

Polar to Cartesian

```
PVector polar( float r, float theta )
{
  float x = r * cos( theta );
  float y = r * sin( theta );

  return new PVector( x, y );
}
```

Polar to Cartesian

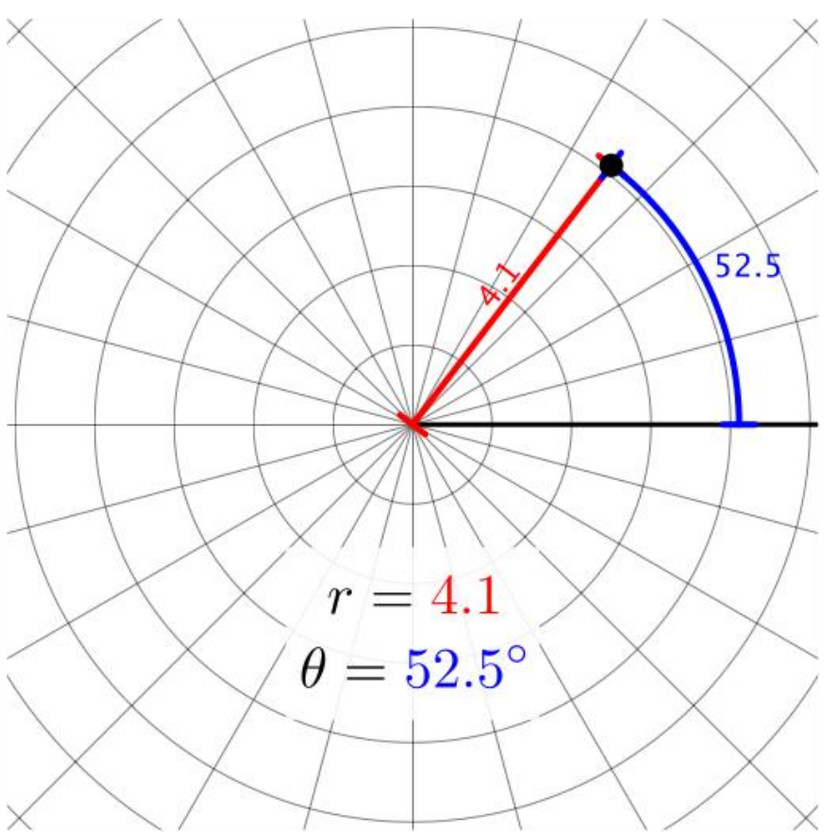
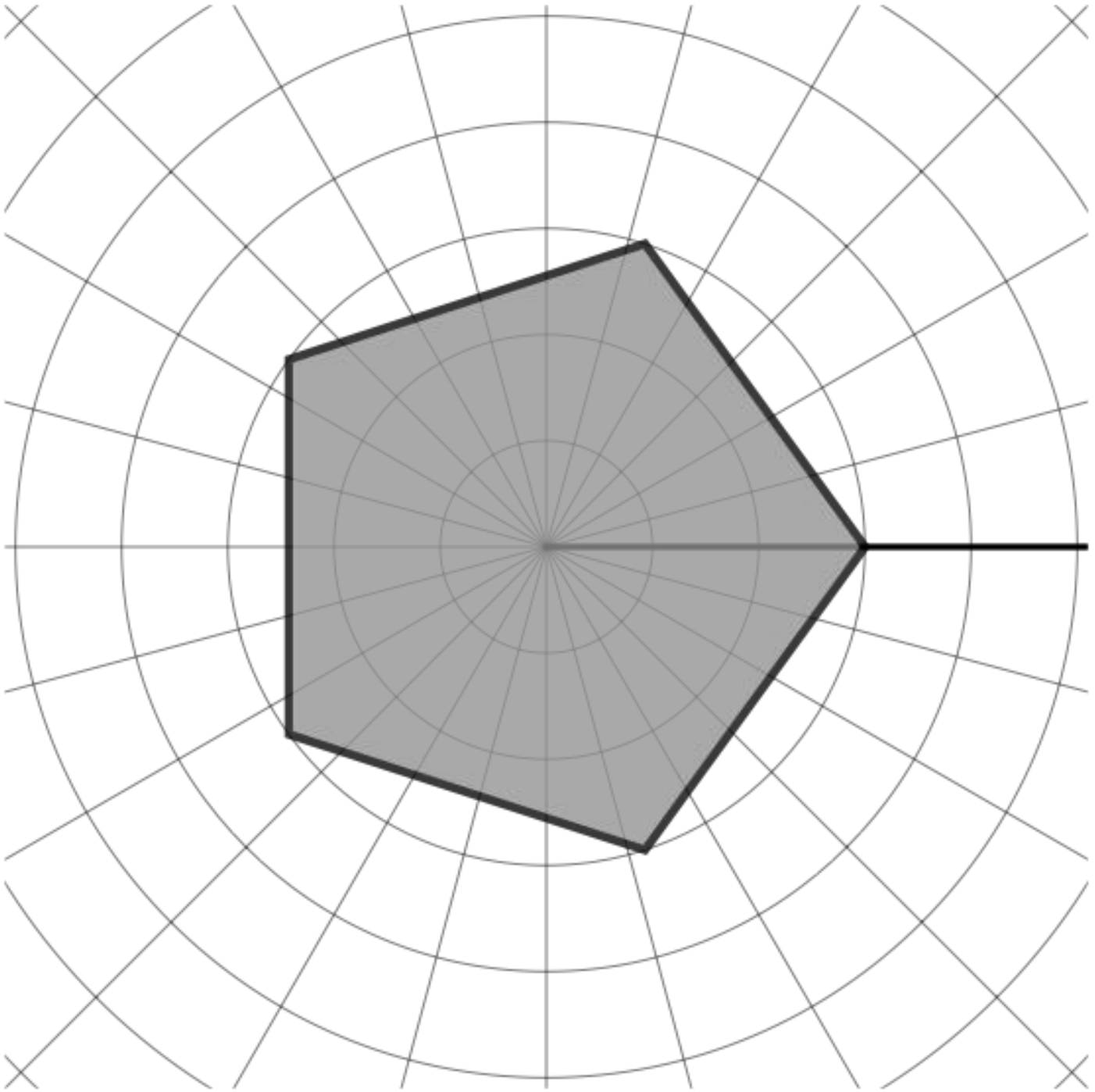
```
PVector polar( float r, float theta )
{
  float x = r * cos( radians( theta ) );
  float y = r * sin( radians( theta ) );

  return new PVector( x, y );
}
```

Let’s specify angles in degrees to avoid headaches.

Polar to Cartesian

This is the part you need to know:

PVector polar(float r, float theta) {…}



Regular polygons