



Module 04

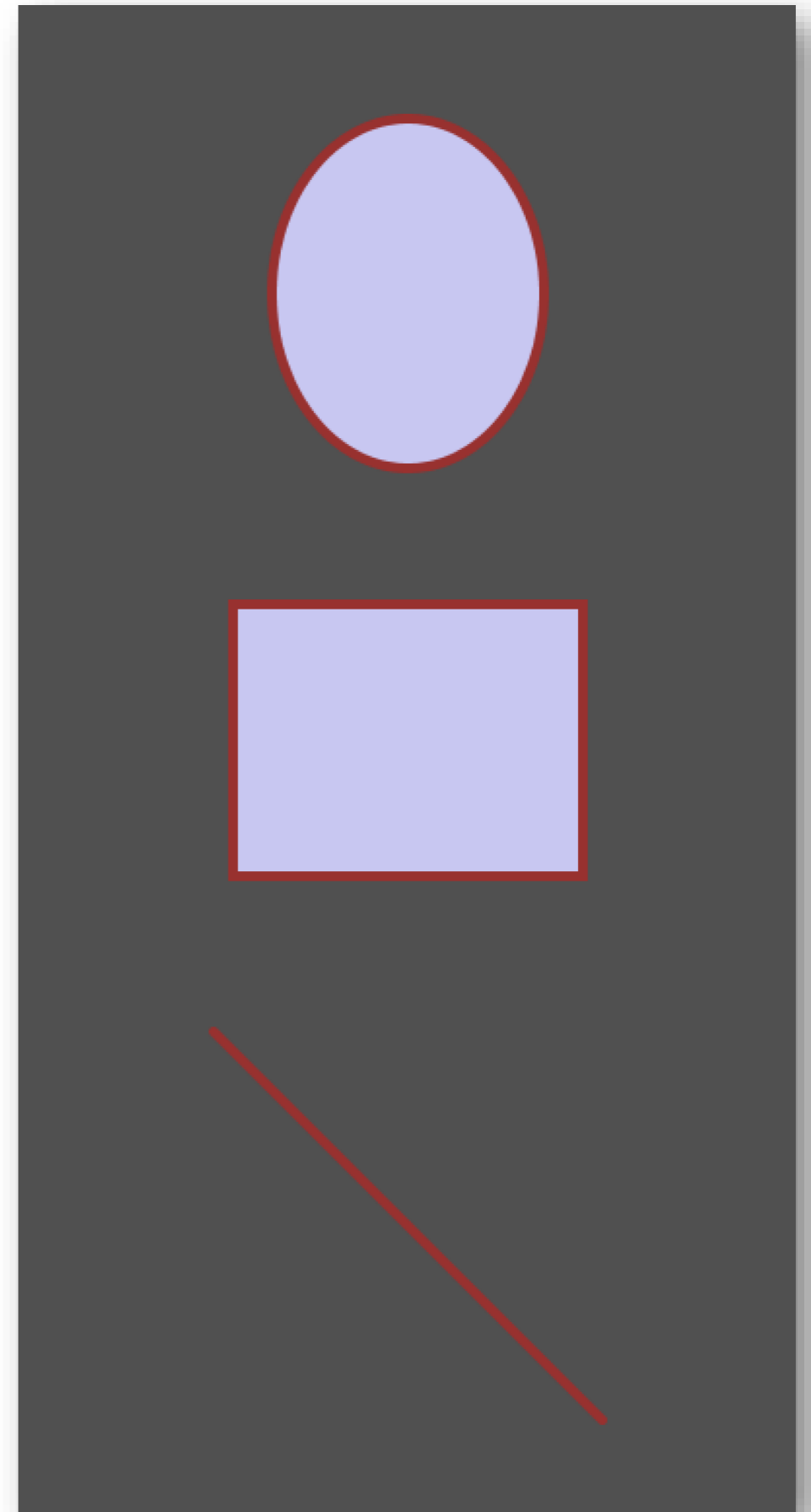
Advanced Shapes

CS 106 Winter 2021

Hollow by Eva Hild

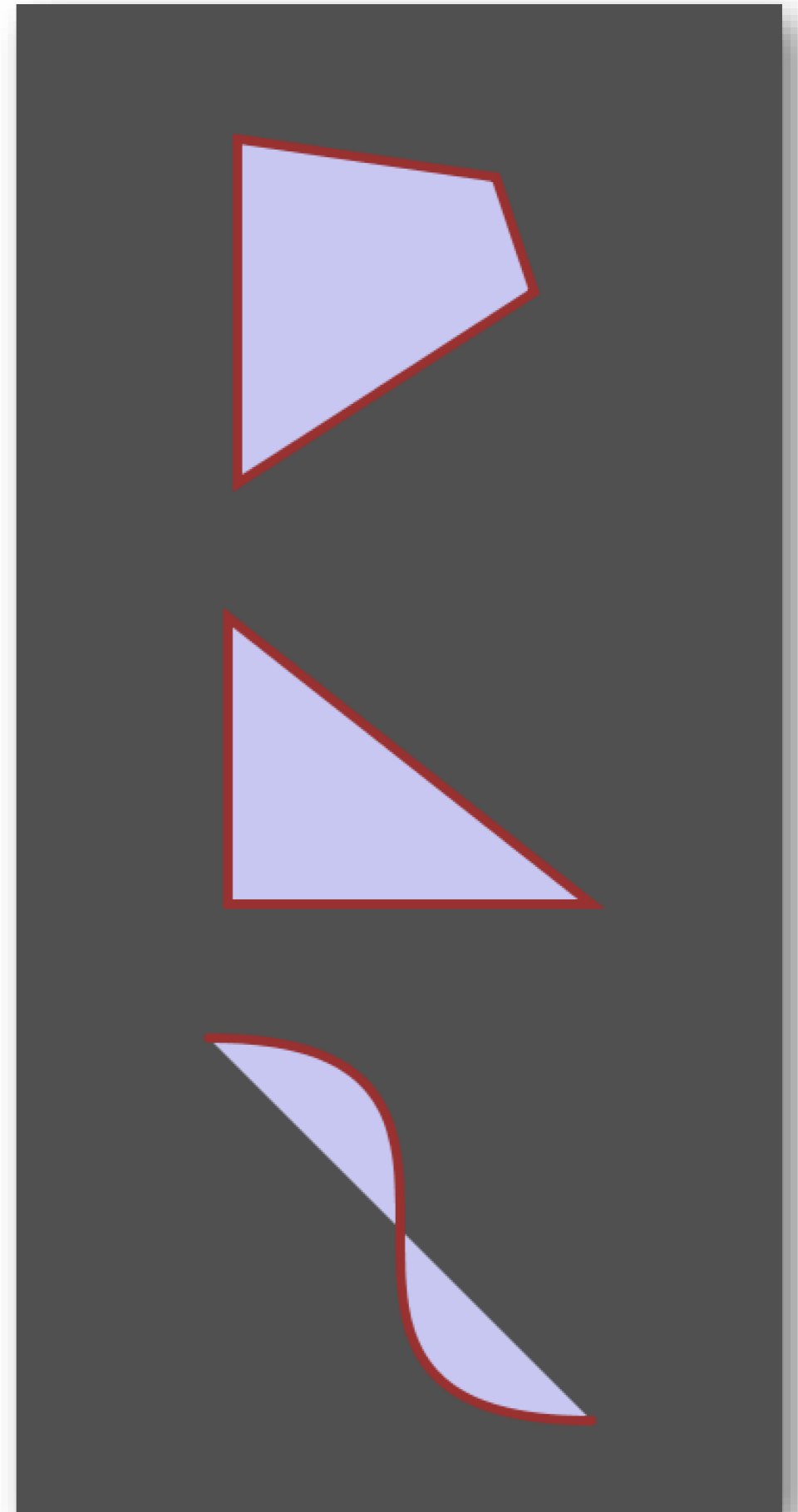
```
function setup() {  
  createCanvas(400, 800);  
  background(80);  
  rectMode(CENTER);  
  
  fill(200, 200, 240);  
  stroke(150, 50, 50);  
  strokeWeight(5);
```

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



```
function setup() {  
  createCanvas(400, 800);  
  background(80);  
  fill(200, 200, 240);  
  stroke(150, 50, 50);  
  strokeWeight(5);
```

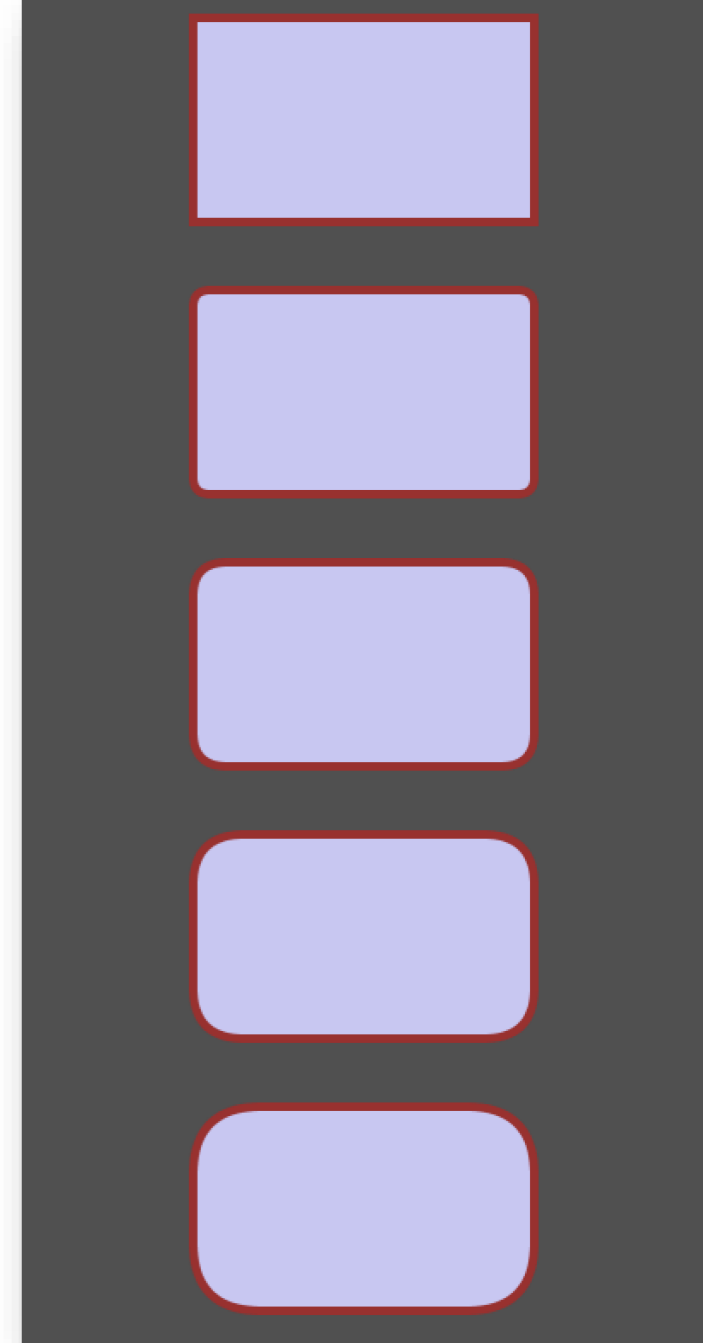
```
  quad(115, 80, 115, 260,  
       270, 160, 250, 100);  
  triangle(110, 330, 110, 480,  
          300, 480);  
  bezier(100, 550, 300, 550,  
        100, 750, 300, 750);  
}
```



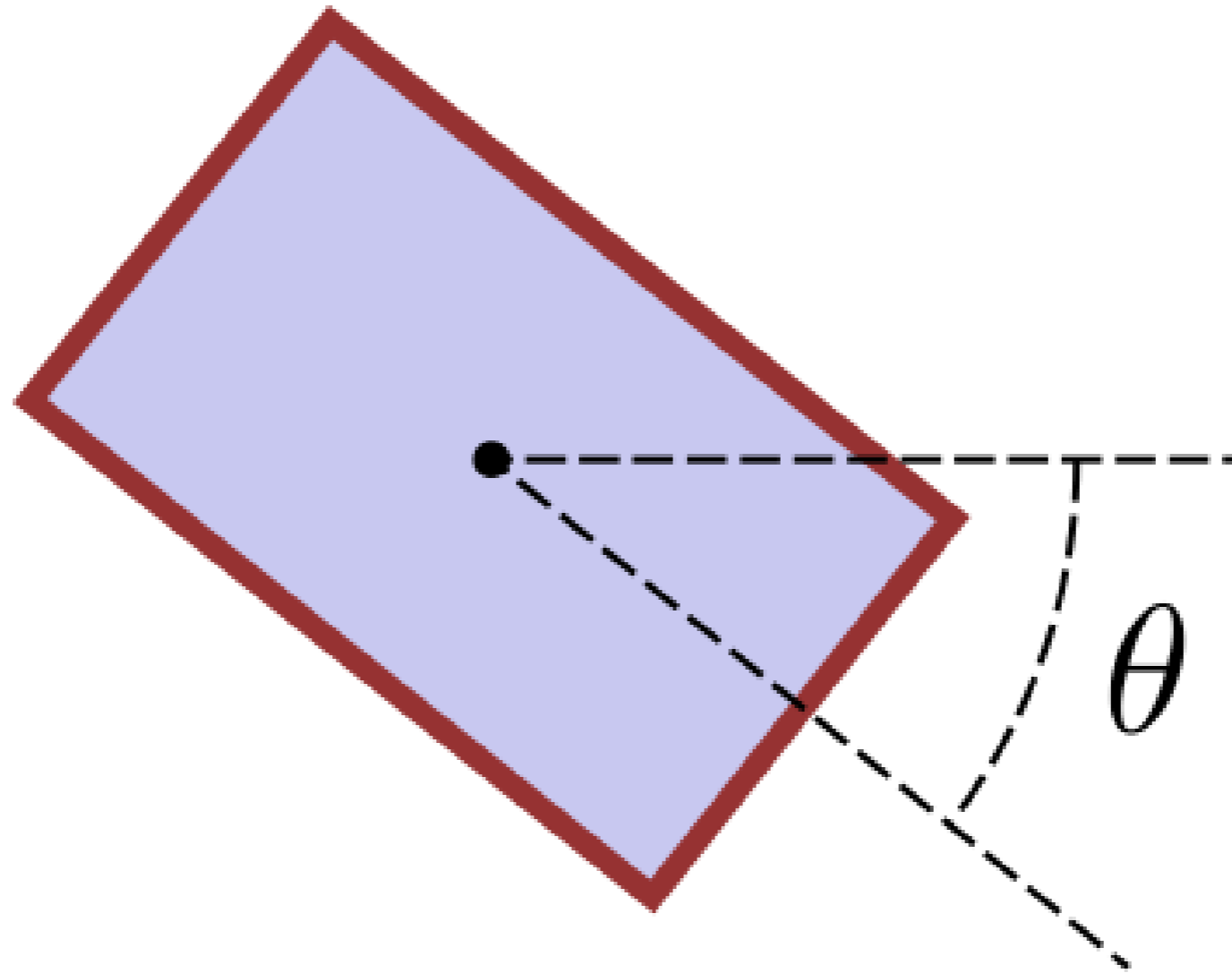
```
function setup() {
  createCanvas(400, 800);
  background(80);

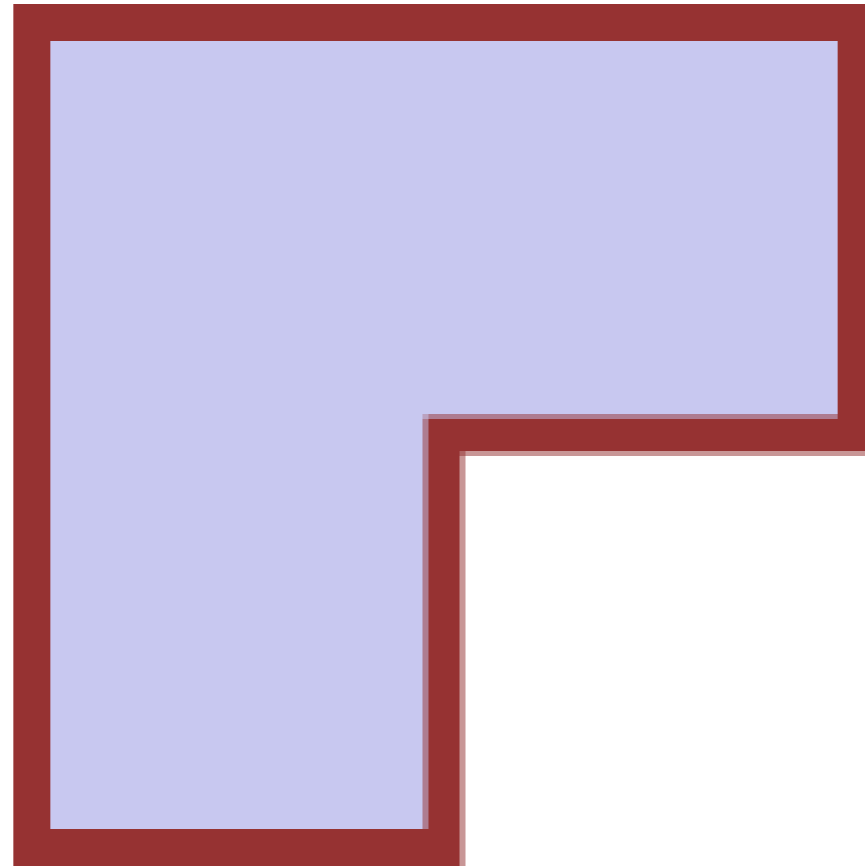
  fill(200, 200, 240);
  stroke(150, 50, 50);
  strokeWeight(5);

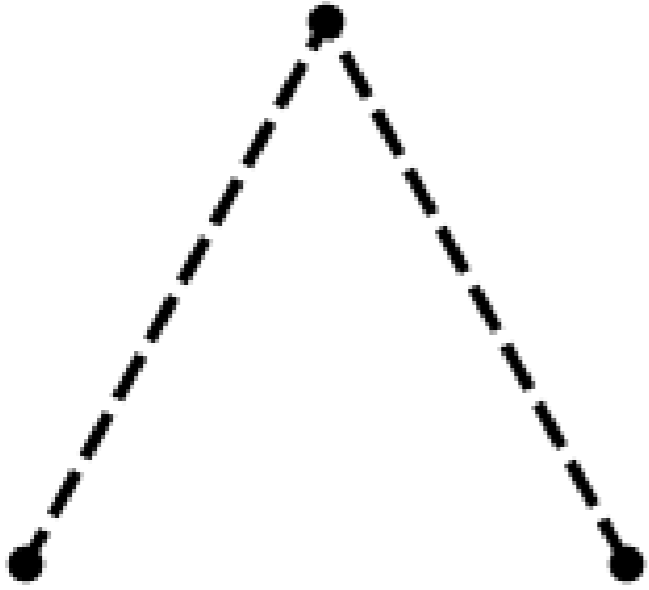
  for (let i = 0; i < 5; i++) {
    rect(100, i * 160 + 20, 200, 120, i * 10);
  }
}
```

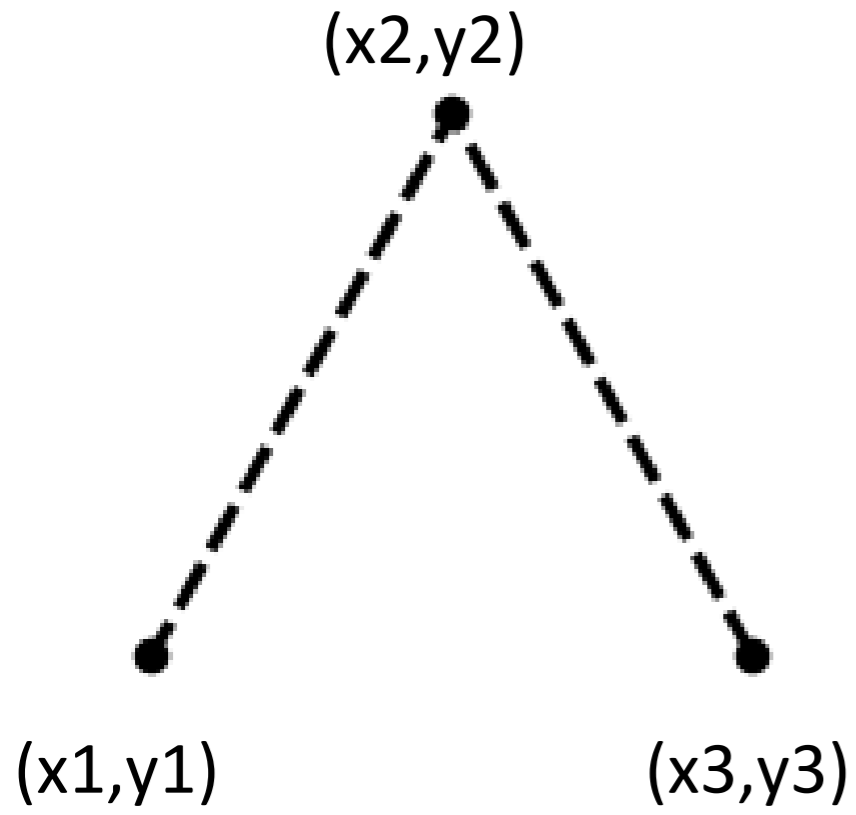


Limitations

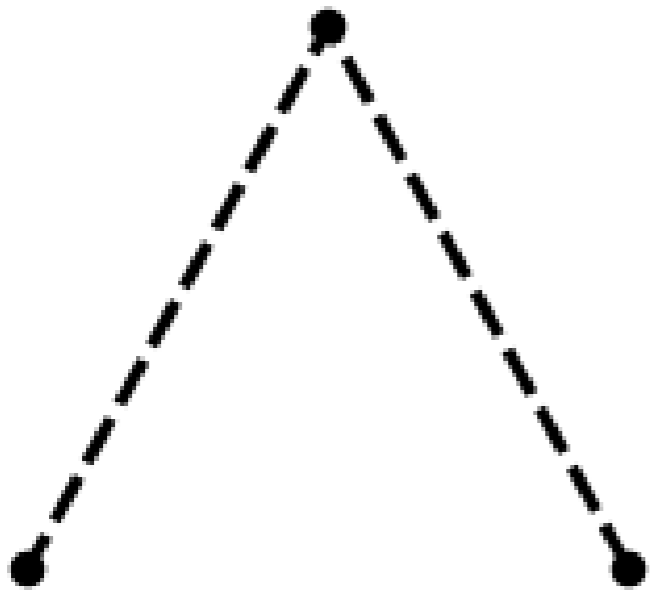








```
line(x1, y1, x2, y2);  
line(x2, y2, x3, y3);
```

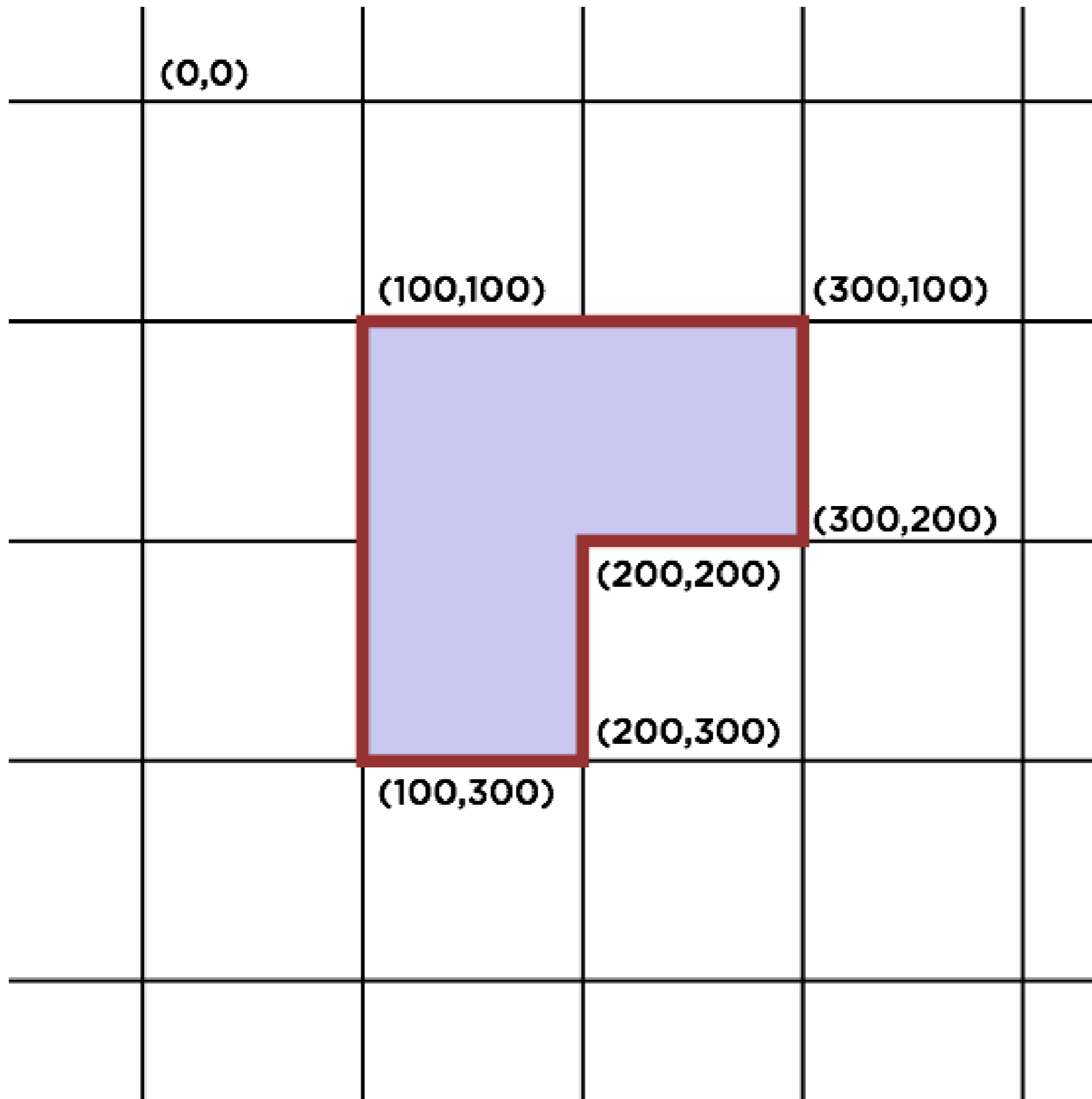
`strokeCap(SQUARE)`



`strokeCap(PROJECT)`



`strokeCap(ROUND)`



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.”

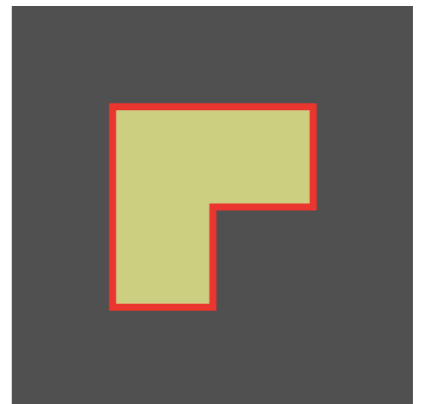
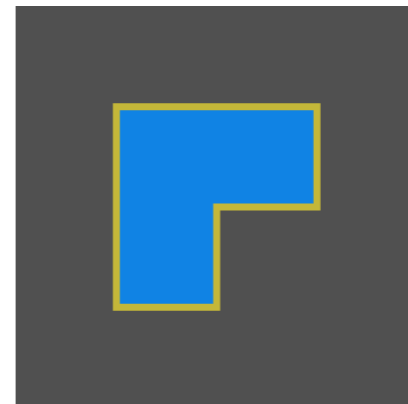
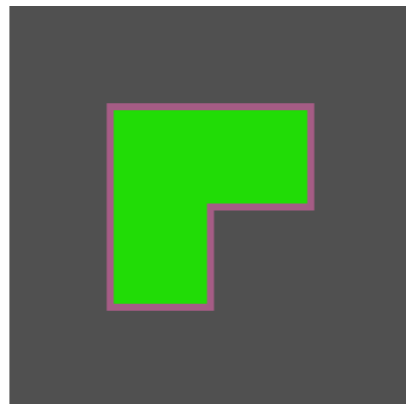
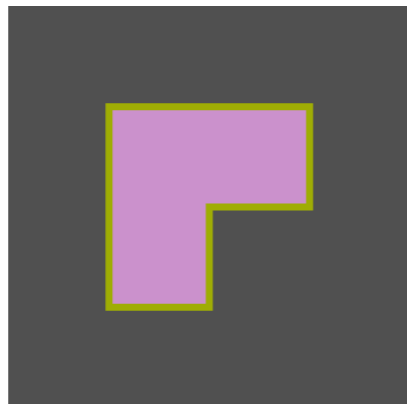
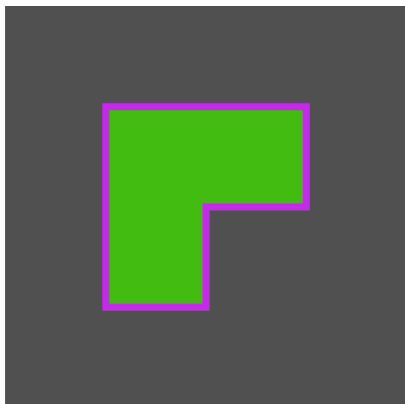
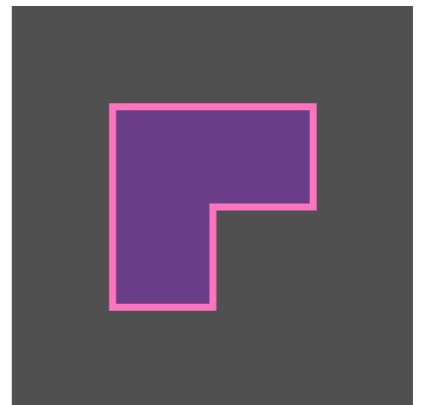
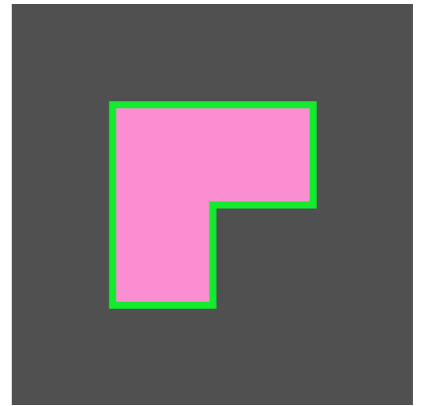
```
function setup() {  
  createCanvas(400, 800);  
  background(80);  
  beginShape();  
    vertex(100, 100);  
    vertex(100, 300);  
    vertex(200, 300);  
    vertex(200, 200);  
    vertex(300, 200);  
    vertex(300, 100);  
  endShape(CLOSE);  
}
```



```
function setup() {
  for (let i = 0; i < 8; i++) {
    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);

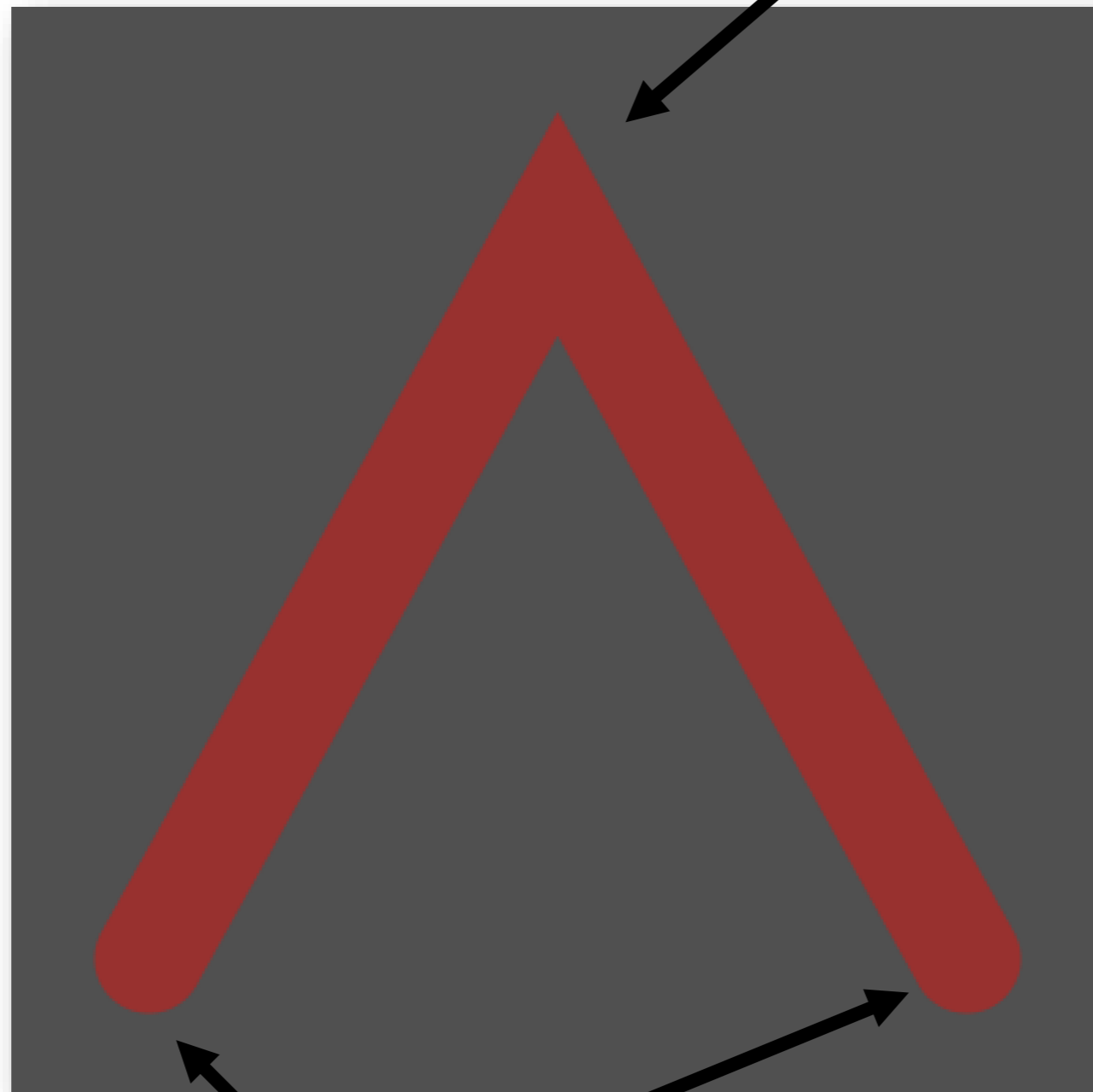
    save("output-" + i + ".png");
  }
}
```



```
function setup() {  
  createCanvas(400, 400);  
  background(80);  
  
  noFill();  
  stroke(150, 50, 50);  
  strokeWeight(40);  
  
  beginShape();  
    vertex(50, 350);  
    vertex(200, 80);  
    vertex(350, 350);  
  endShape();  
}
```

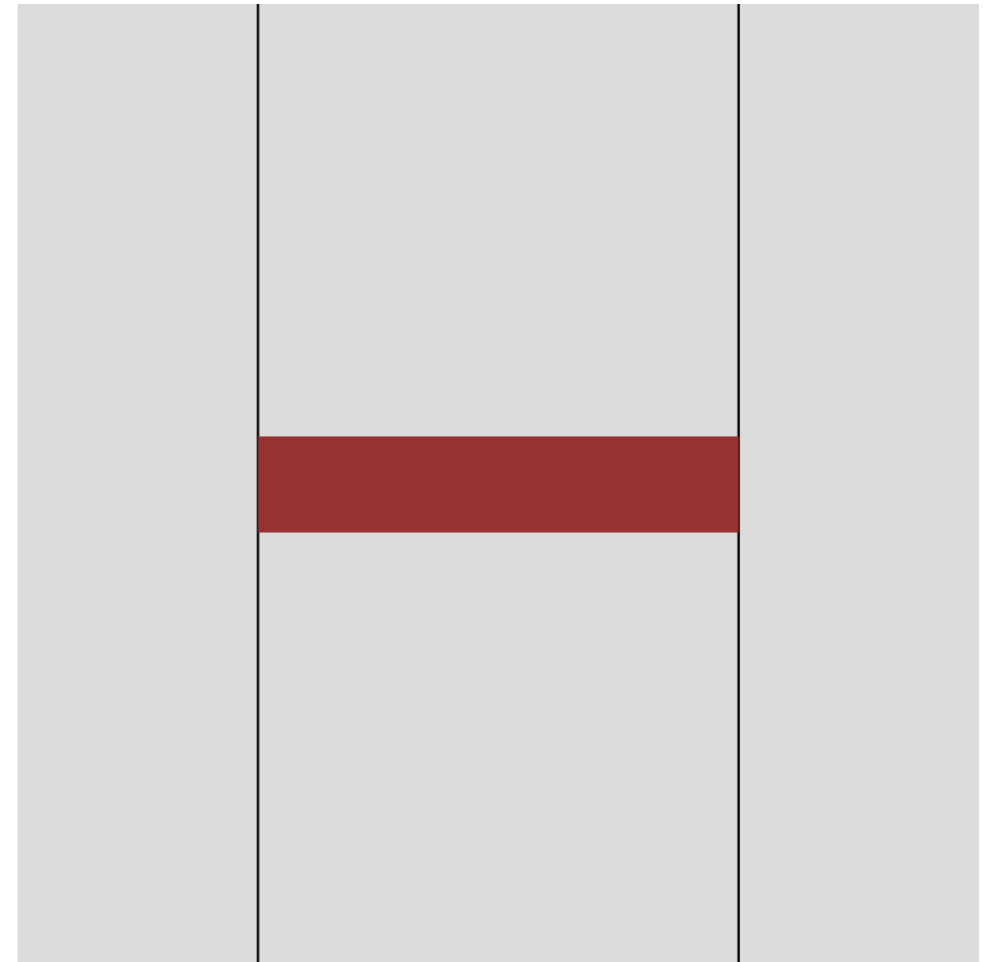


`strokeJoin()`

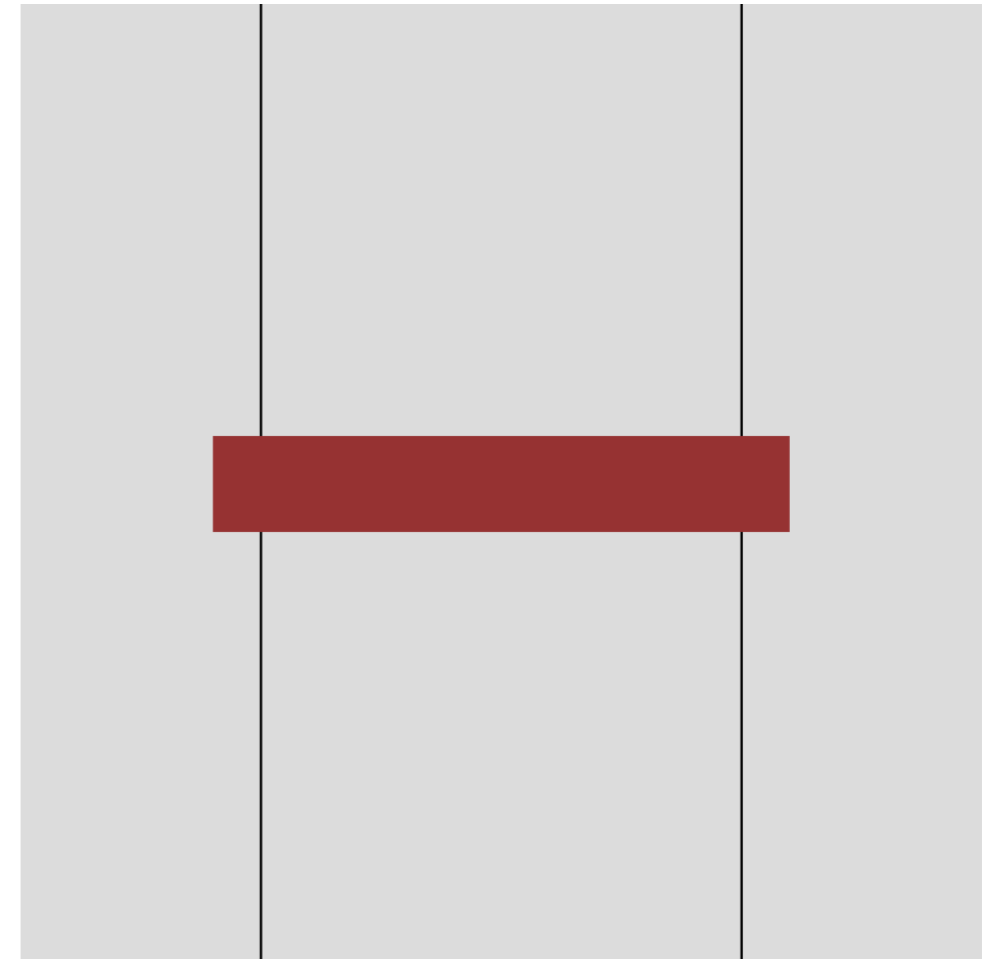


`strokeCap()`

```
function setup() {  
  createCanvas(400, 400);  
  background(220);  
  
  strokeCap(SQUARE);  
  line(100, 0, 100, height);  
  line(300, 0, 300, height)  
  
  stroke(150, 50, 50);  
  strokeWeight(40);  
  beginShape();  
  vertex(100, 200);  
  vertex(300, 200);  
  endShape();  
}
```

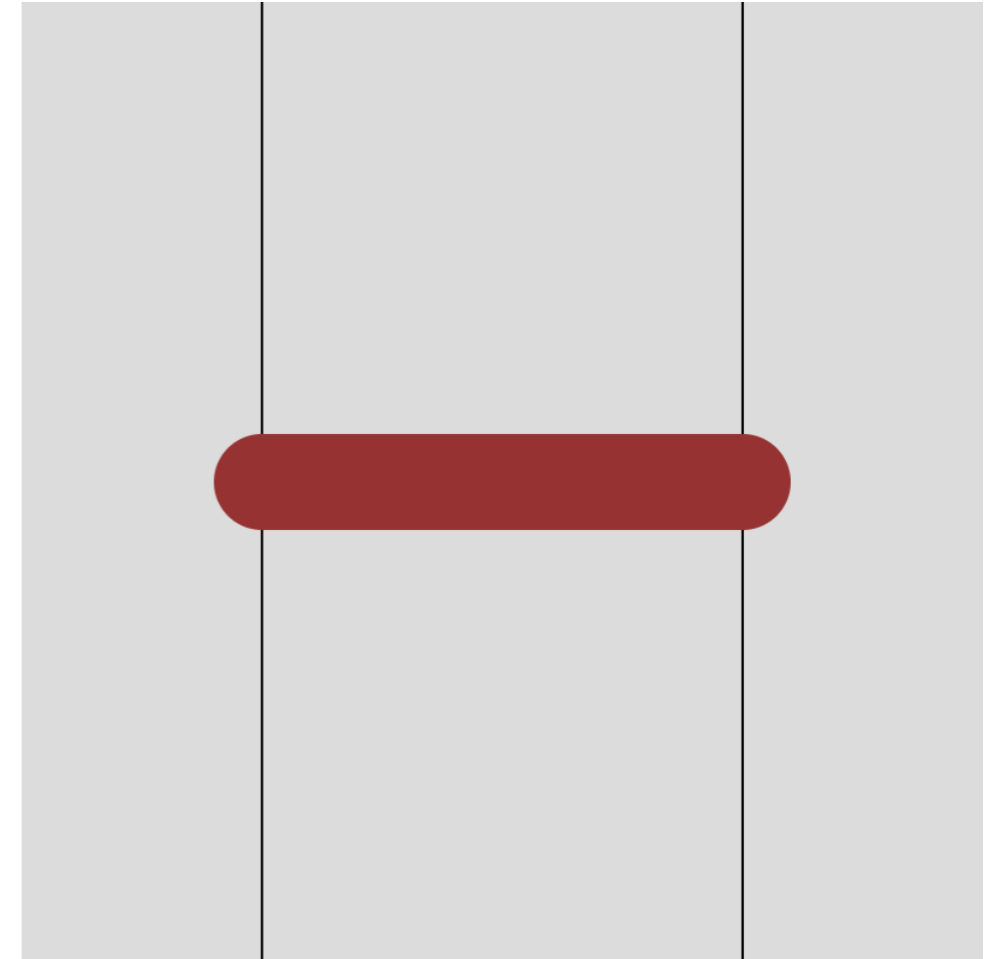


```
function setup() {  
  createCanvas(400, 400);  
  background(220);  
  
  strokeCap(PROJECT);  
  line(100, 0, 100, height);  
  line(300, 0, 300, height)  
  
  stroke(150, 50, 50);  
  strokeWeight(40);  
  beginShape();  
  vertex(100, 200);  
  vertex(300, 200);  
  endShape();  
}
```

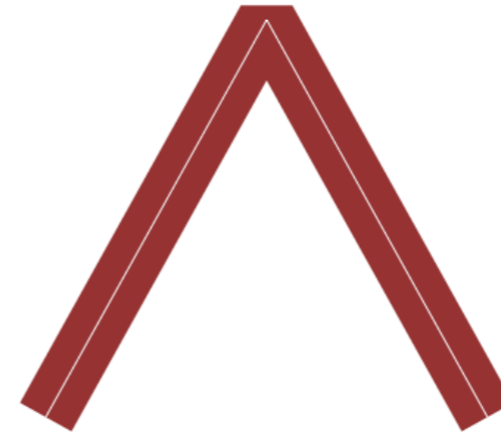


ROUND is the default

```
function setup() {  
  createCanvas(400, 400);  
  background(220);  
  
  strokeCap(ROUND);  
  line(100, 0, 100, height);  
  line(300, 0, 300, height)  
  
  stroke(150, 50, 50);  
  strokeWeight(40);  
  beginShape();  
  vertex(100, 200);  
  vertex(300, 200);  
  endShape();  
}
```



strokeCap (SQUARE)



strokeCap (PROJECT)



strokeCap (ROUND)



strokeJoin (MITER)

strokeJoin (BEVEL)

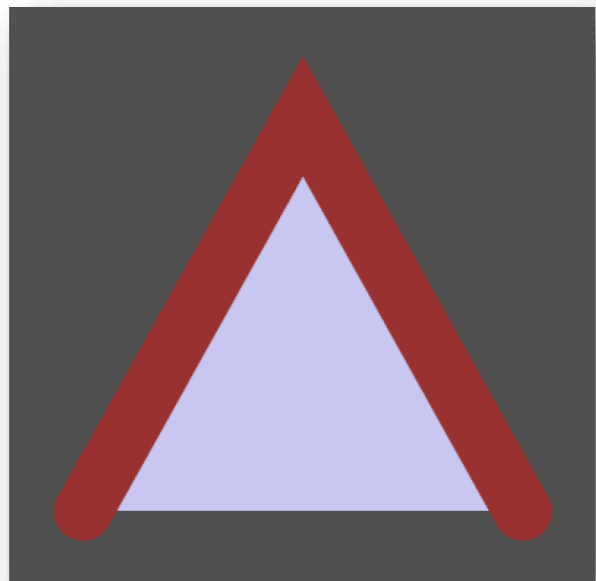
strokeJoin (ROUND)

Try combinations of `strokeCap()` and `strokeJoin()`

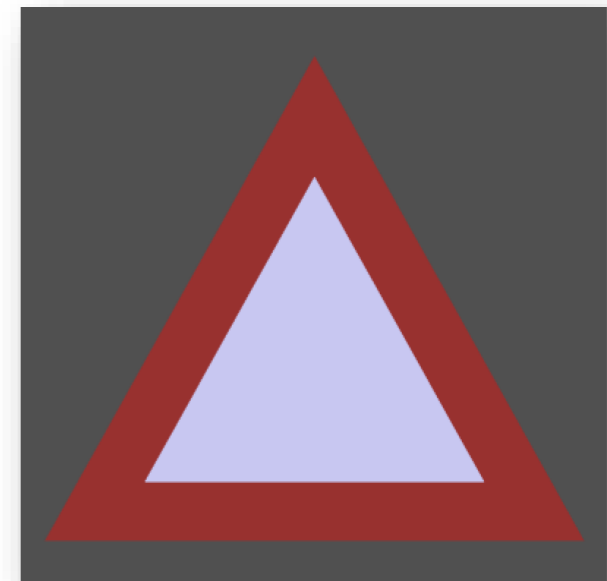
```
function setup() {  
  createCanvas(400, 400);  
  background(220);  
  noFill();  
  stroke(150, 50, 50);  
  strokeWeight(40);  
  
  strokeCap(ROUND);  
  strokeJoin(MITER);  
  
  beginShape();  
  vertex(100, 300);  
  vertex(200, 100);  
  vertex(300, 300);  
  endShape();  
}
```

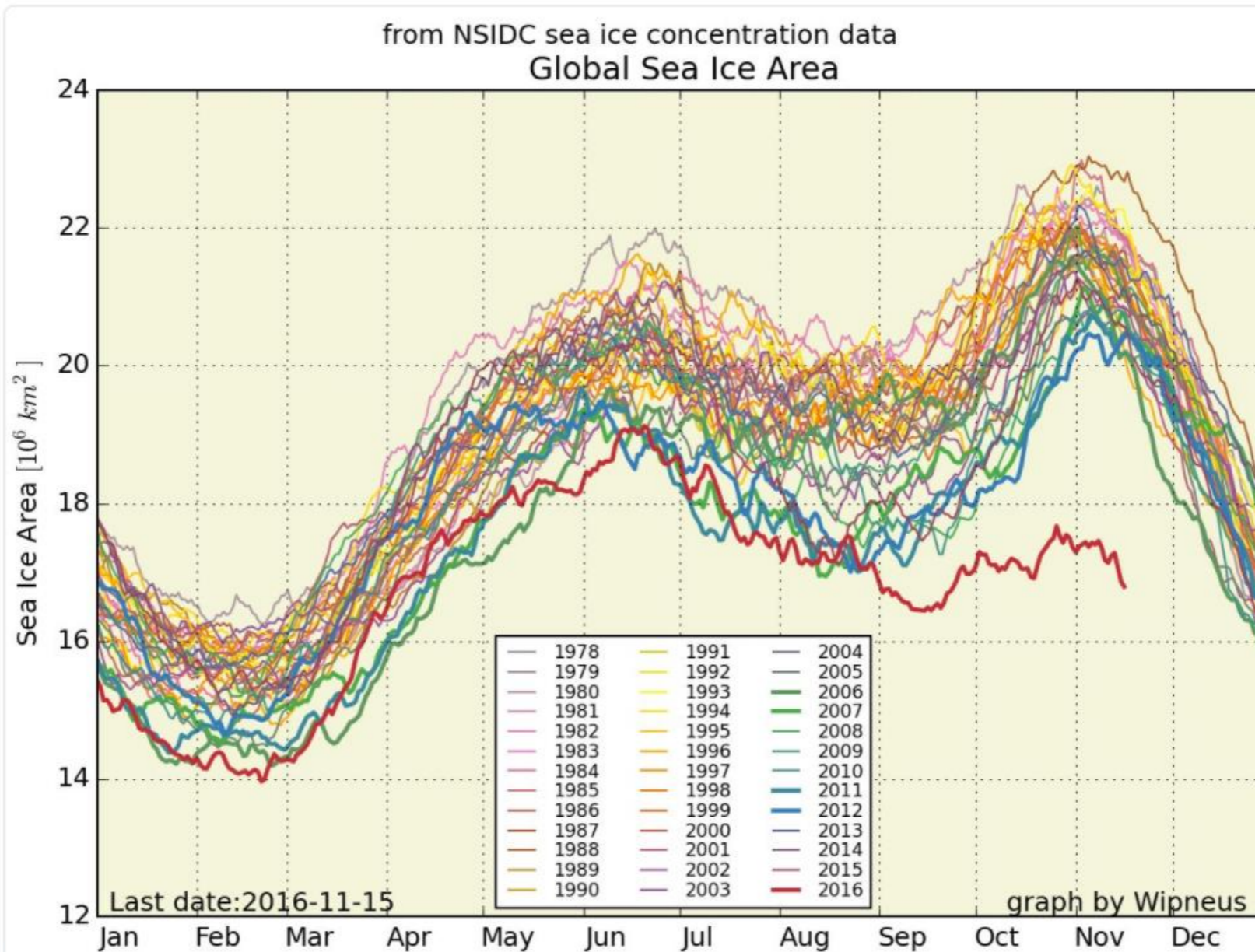


```
function setup() {  
  createCanvas(400, 400);  
  background(80);  
  fill(200, 200, 240);  
  stroke(150, 50, 50);  
  strokeWeight(40);  
  
  beginShape();  
  vertex(50, 350);  
  vertex(200, 80);  
  vertex(350, 350);  
  endShape();  
}
```



```
function setup() {  
  createCanvas(400, 400);  
  background(80);  
  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...](#)

[Animate Monthly Images](#) [Compare Anomalies](#) [Compare Trends](#) [Documentation](#) [FAQ](#) [Data Archive](#)

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://sidacs.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

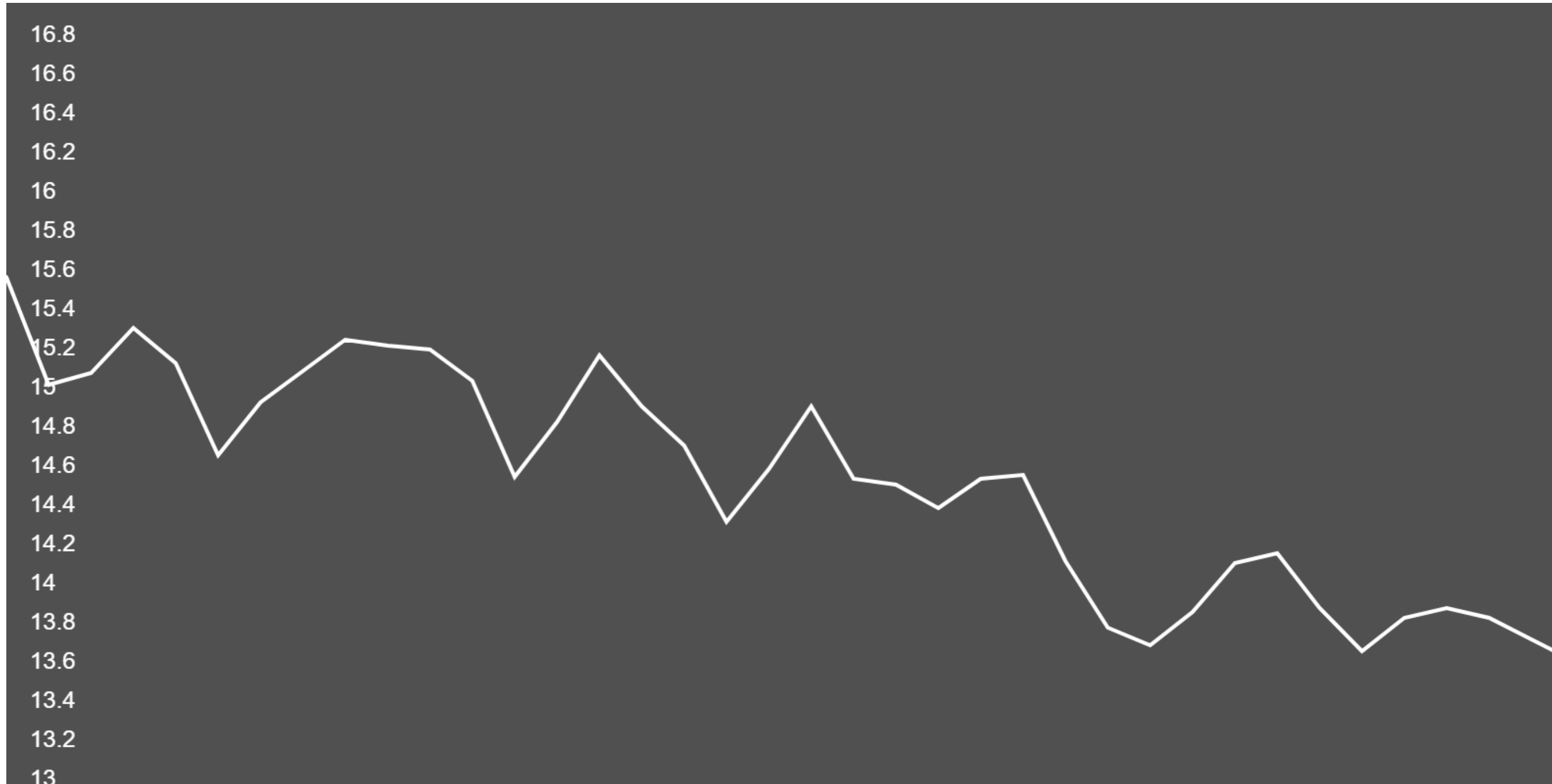
15.60
15.05
15.11
15.34
15.16
14.69
14.96
15.12
15.28
15.25
15.23
15.07
14.58
14.86
15.20
14.94
14.74
14.35
14.62
14.94
14.57

Data for January only for all years (1979 at the top of the file to 2005 at the bottom of the file.

Notice that the numbers are generally decreasing (so our graph should be decreasing).

File is "january.txt".

We will create a graph as shown here.
We plot the January data from 1979-2015



Open Processing: Week4_Sea_Ice_January (slide 1 of 2)

<https://openprocessing.org/sketch/1077561>

```
let samps;

function preload() {
  samps = loadStrings("january.txt");
}

function setup() {
  createCanvas(800, 400);

  // We looked at the january.txt file before writing our program.
  // We saw that the maximum was 15.6 and minimum was 13.64. So later
  // in the code in map() we say the numbers are between 13-17.

  background(80);
  stroke(255);
  strokeWeight(2);
  noFill();
```

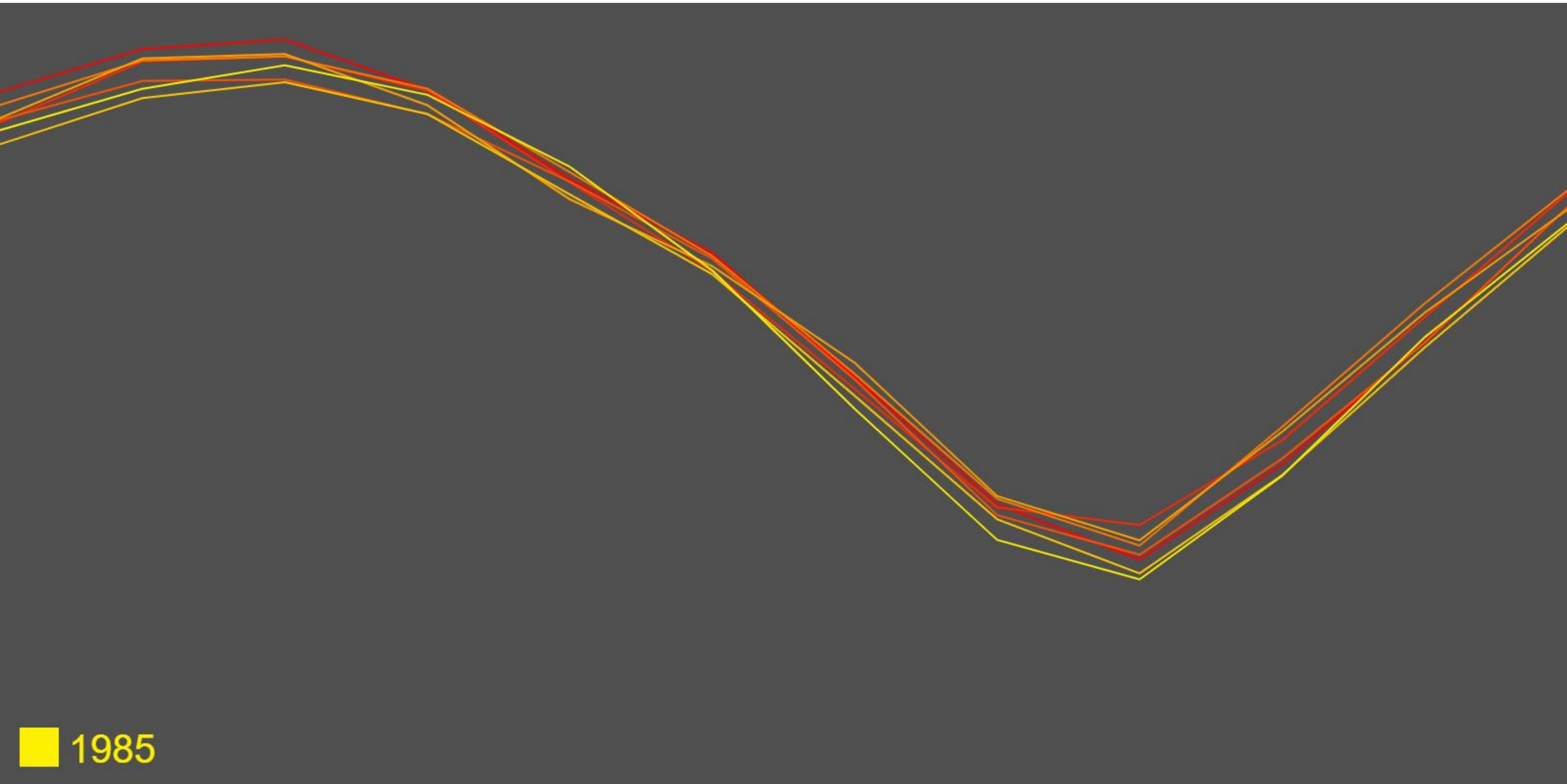
Open Processing: Week4_Sea_Ice_January (slide 2 of 2)

<https://openprocessing.org/sketch/1077561>

```
beginShape();
for (let idx = 0; idx < samps.length; ++idx) {
  vertex(
    map(idx, 0, samps.length - 1, 0, width),
    map(samps[idx], 13, 17, height, 0));
}
endShape();

for (let idx = 0; idx < 21; ++idx) {
  let y = map(idx, 0, 20, height, 0);
  noStroke();
  fill(255);
  text("" + map(idx, 0, 20, 13, 17), 12, y);
}
}
```

We will create a graph as shown here.
We plot the monthly data from 1979-2015



Open Processing: Week4_Sea_Ice_Monthly (slide 1 of 4)

<https://openprocessing.org/sketch/1077652>

```
let lines;
let cur;

function preload() {
  lines = loadStrings("monthly.txt");
}

function setup() {
  createCanvas(800, 400);
  cur = 0;
  textSize(20);
}
```

Open Processing: Week4_Sea_Ice_Monthly (slide 2 of 4)

<https://openprocessing.org/sketch/1077652>

```
function draw() {
  background(80);

  colorMode(HSB, 255);
  noFill();
  strokeWeight(1);

  for (let idx = 0; idx < cur; ++idx) {
    stroke(map(idx, 0, lines.length - 1, 0, 255), 255, 255);
    drawGraph(lines[idx]);
  }

  if (cur > 0) {
    noStroke();
    fill(map(cur - 1, 0, lines.length - 1, 0, 255), 255, 255);
    rect(10, height - 30, 20, 20);
    text(splitTokens(lines[cur - 1])[0], 35, height - 12);
  }
  frameRate(1);
}
```

Open Processing: Week4_Sea_Ice_Monthly (slide 3 of 4)

<https://openprocessing.org/sketch/1077652>

```
function drawGraph(line) {
  let vals = float(splitTokens(line));
  vals = vals.slice(1, 13);
  // println( min( vals ), max( vals ) );
  beginShape();
  for (let idx = 0; idx < vals.length; ++idx) {
    let x = map(idx, 0, vals.length - 1, 0, width);
    let y = map(vals[idx], 3, 17, height, 0);
    vertex(x, y);
  }
  endShape();
}
```

Open Processing: Week4_Sea_Ice_Monthly (slide 4 of 4)

<https://openprocessing.org/sketch/1077652>

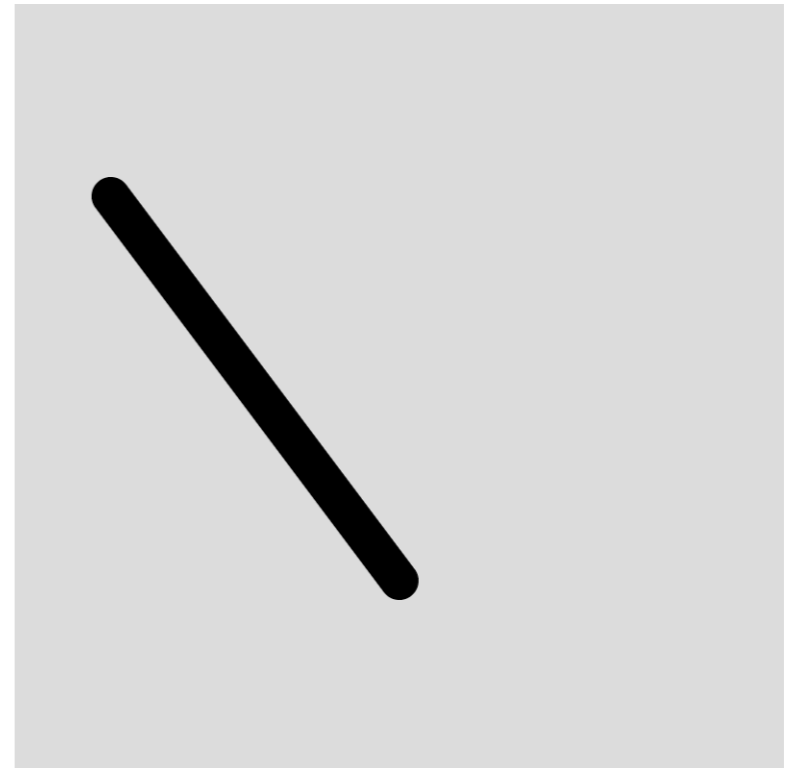
```
function keyPressed() {  
  if (keyCode == LEFT_ARROW) {  
    cur = max(cur - 1, 0);  
  } else if (keyCode == RIGHT_ARROW) {  
    cur = min(cur + 1, lines.length);  
  }  
}
```


Representing Points `createVector()`

- Represents a point
- You may feel that “`createVector`” is a highly unusual name for something that represents a point. If so, we agree with you.
- It is an object (sometimes called a container)
- It has `x` and `y` properties

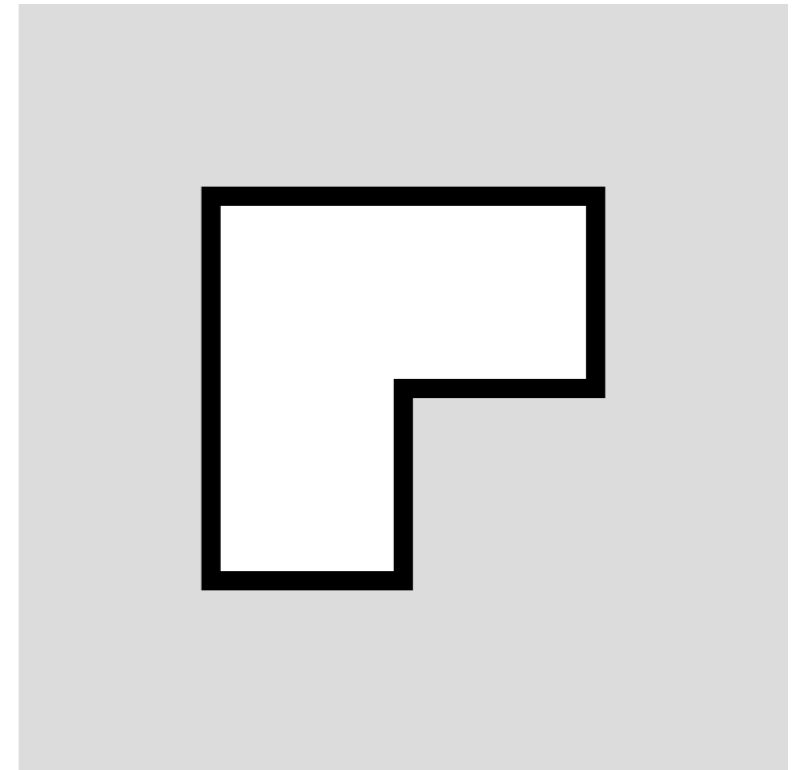
Representing Points createVector()

```
function setup() {  
  createCanvas(400, 400);  
  background(220);  
  strokeWeight(20);  
  
  let pt1 = createVector(50, 100);  
  let pt2 = createVector(200, 300);  
  
  line(pt1.x, pt1.y, pt2.x, pt2.y);  
}
```



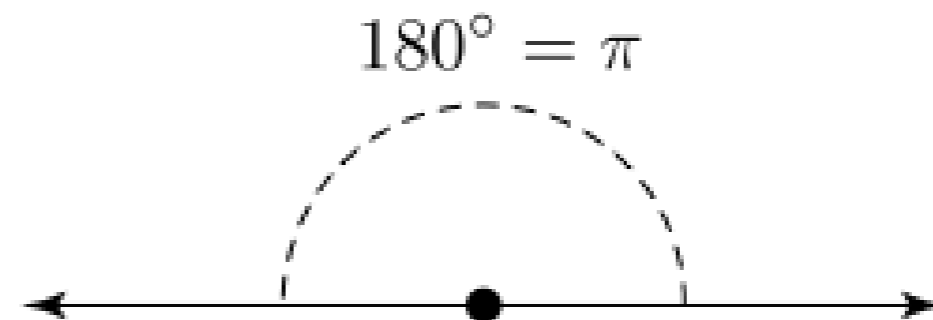
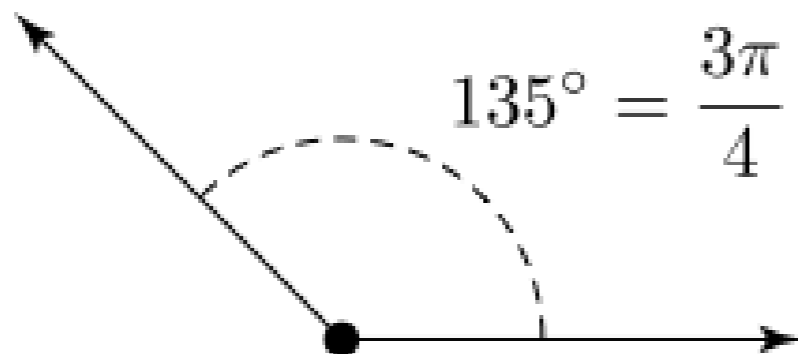
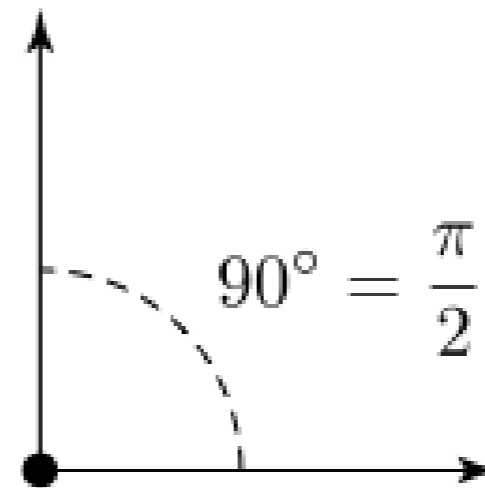
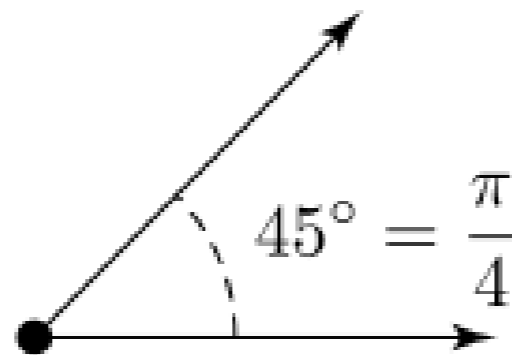
An array using createVector()

```
function setup() {  
  createCanvas(400, 400);  
  background(220);  
  strokeWeight(10);  
  
  let pts = [  
    createVector(100, 100),  
    createVector(100, 300),  
    createVector(200, 300),  
    createVector(200, 200),  
    createVector(300, 200),  
    createVector(300, 100)  
  ];  
  
  beginShape();  
  for (let i = 0; i < pts.length; i++) {  
    vertex(pts[i].x, pts[i].y);  
  }  
  endShape(CLOSE);  
}
```

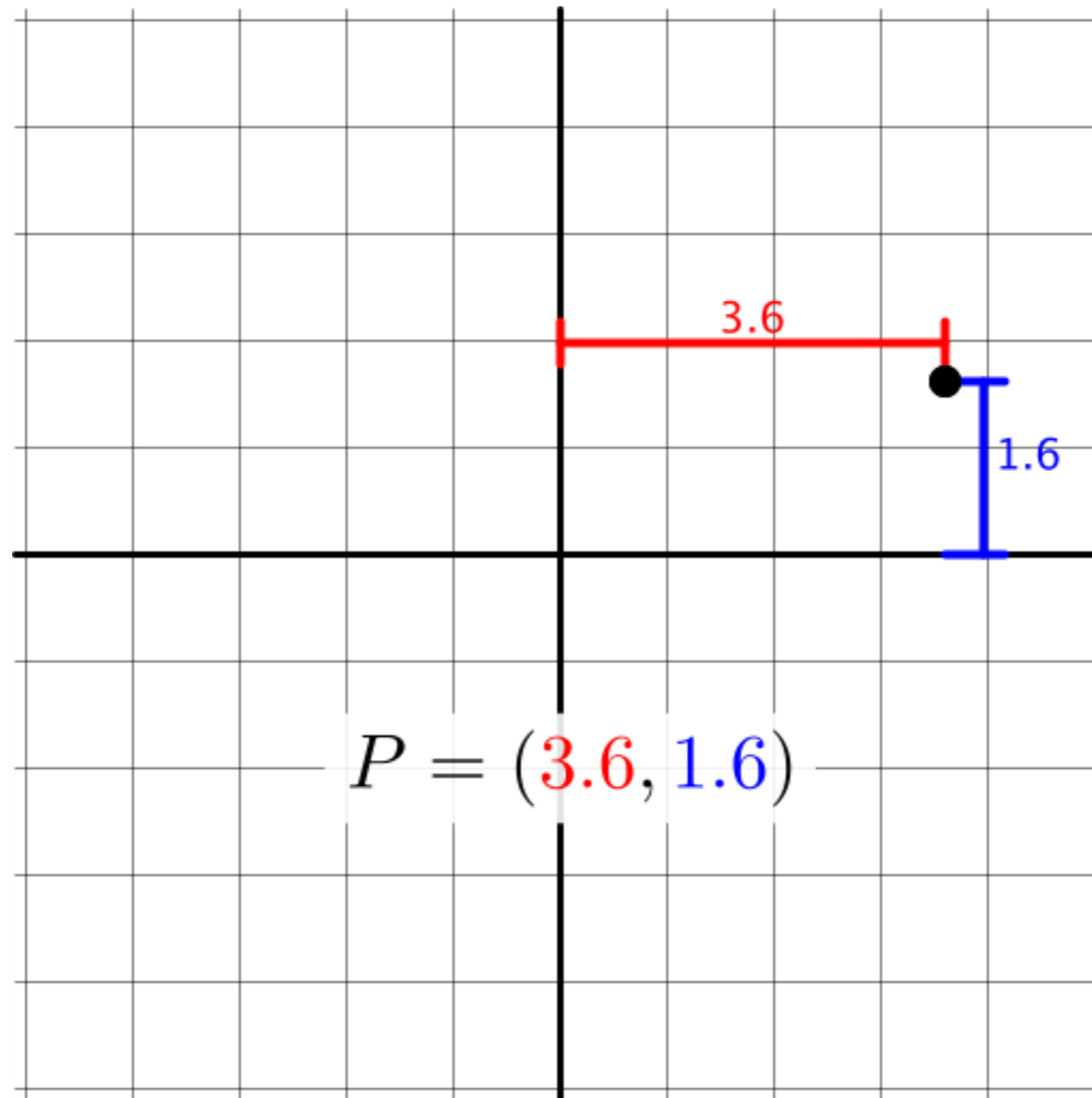


Angles

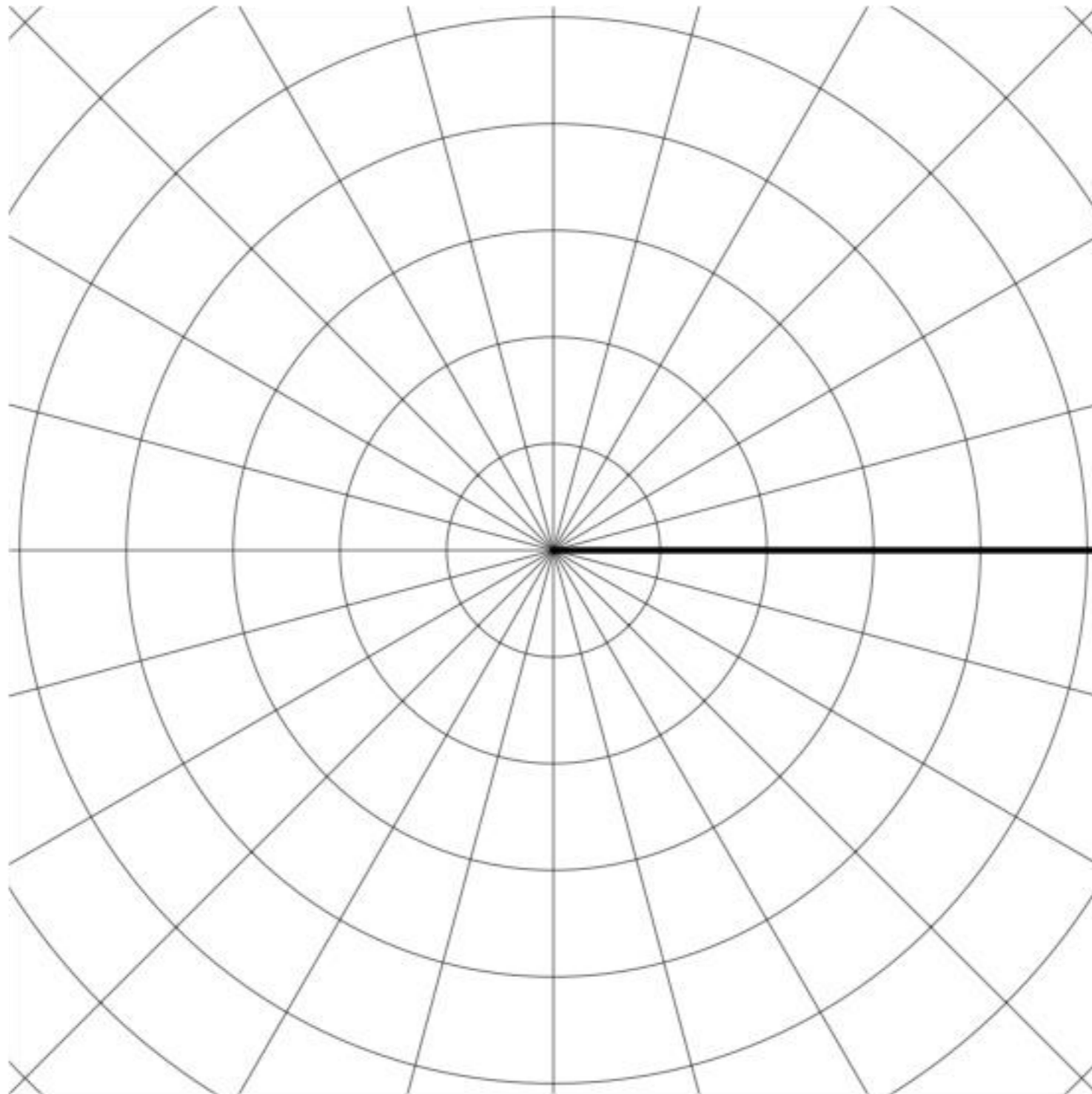
An *angle* is the measurement of the opening between two lines. We learn to measure angles in degrees, but programming languages usually expect them in *radians*.



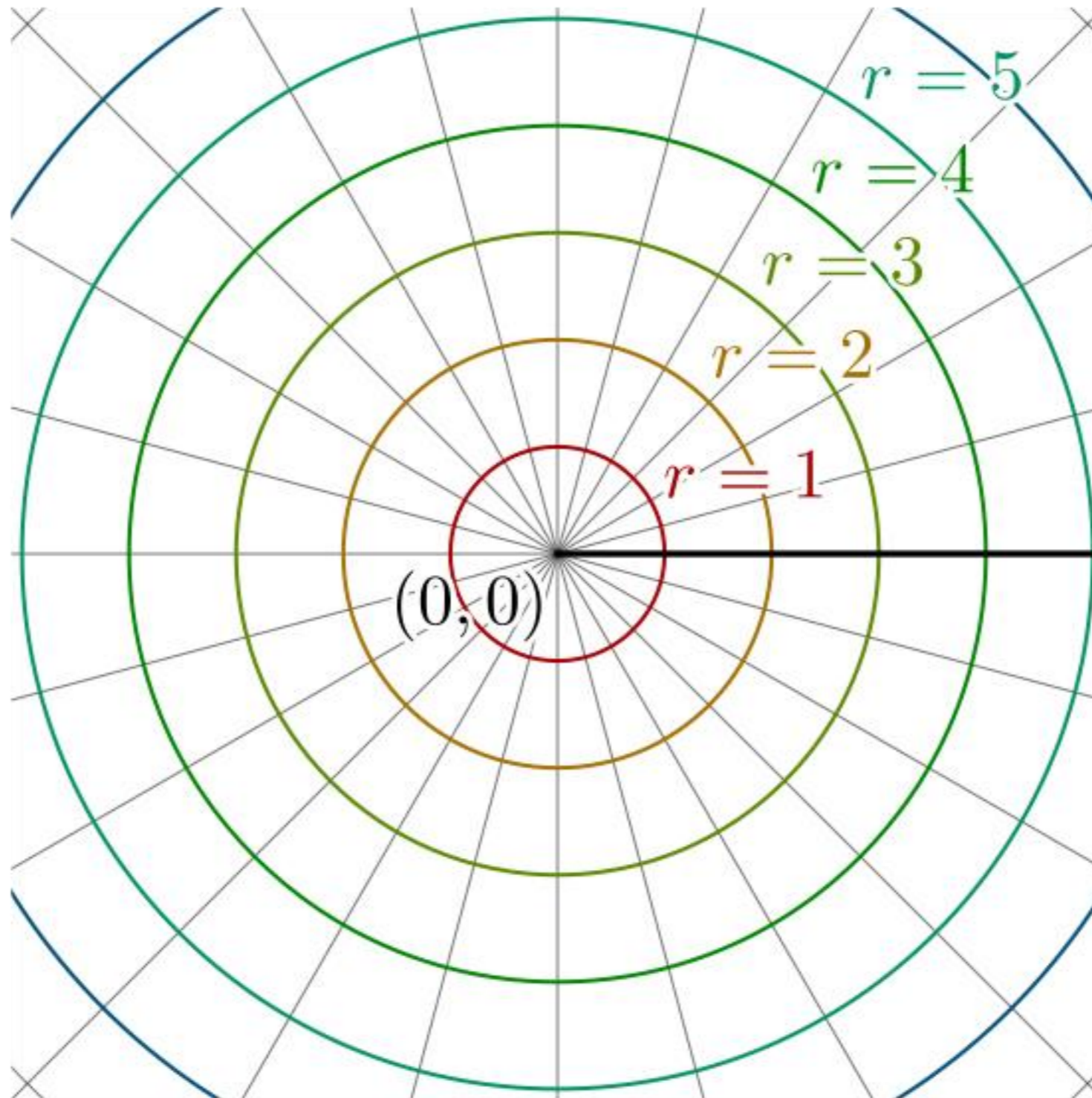
Cartesian (i.e., regular) coordinates



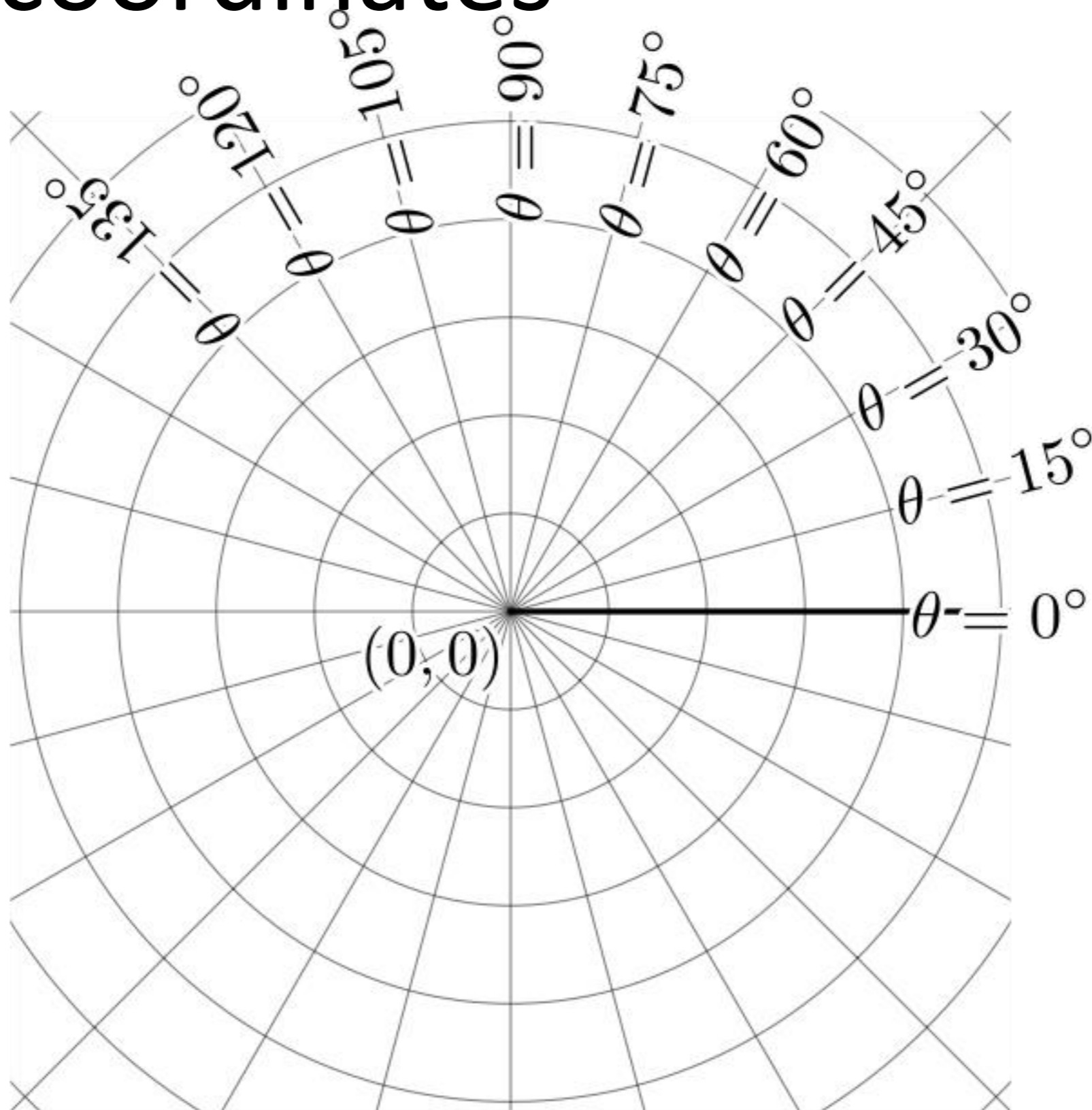
Polar coordinates



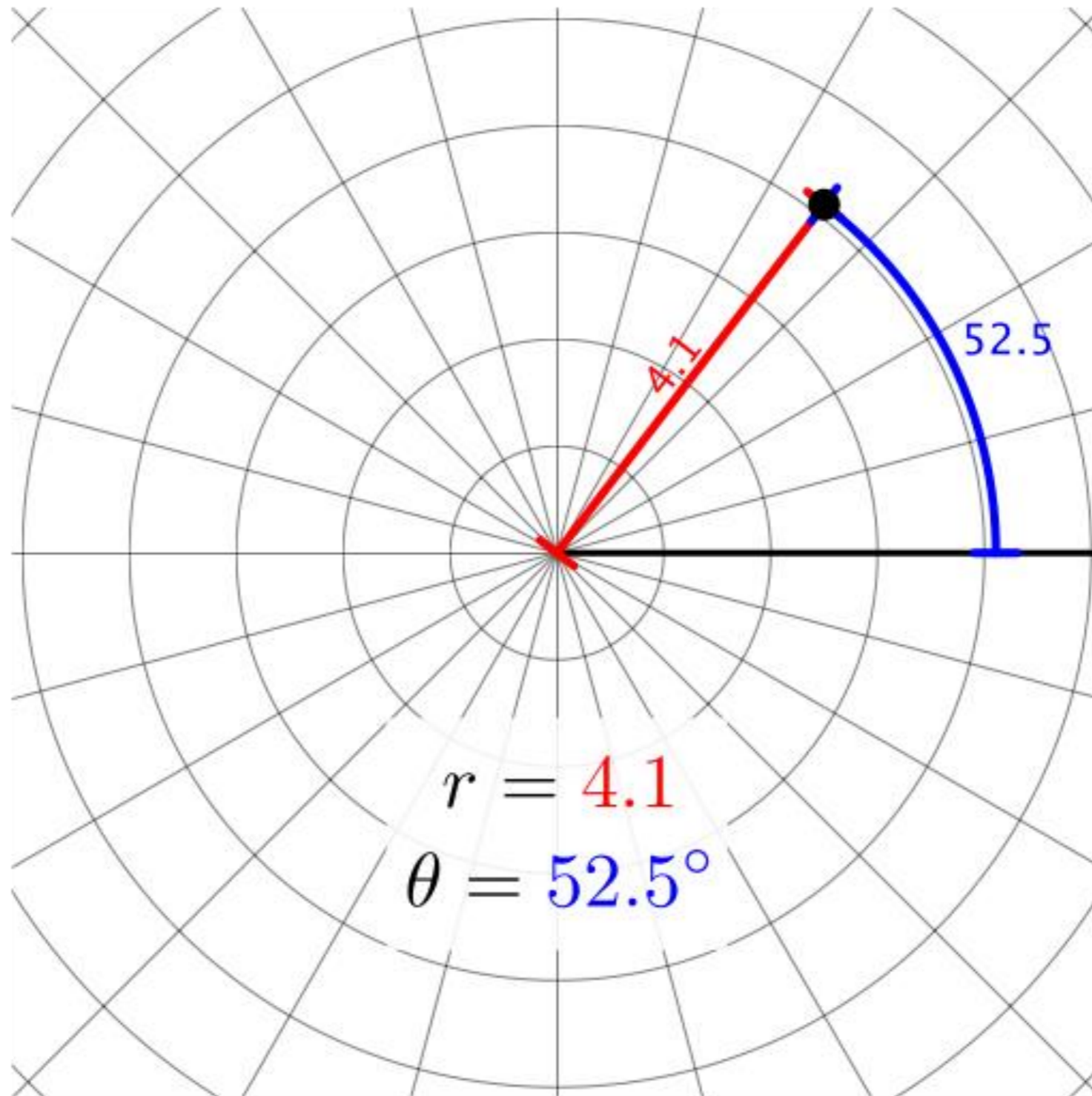
Polar coordinates



Polar coordinates

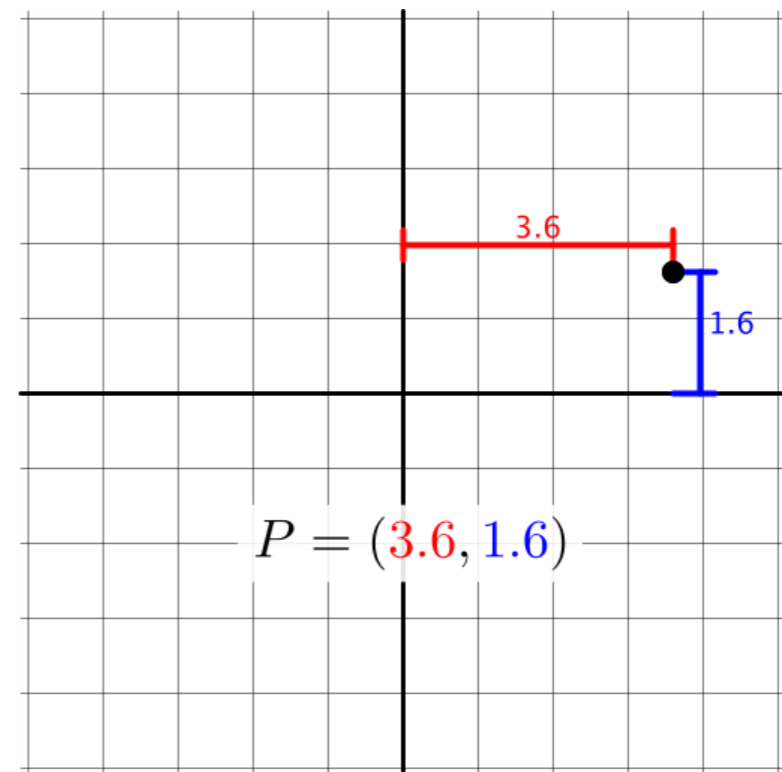
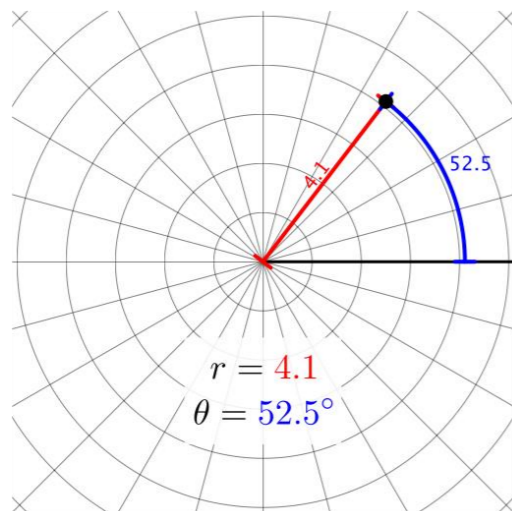


Polar coordinates



Polar to Cartesian

function `polar(r, theta)` {...}



Our uWaterloo Polar function

```
function polar(r, theta) {  
  let dt = radians(theta);  
  
  let x = r * cos(dt);  
  let y = r * sin(dt);  
  
  return createVector(x, y);  
}
```

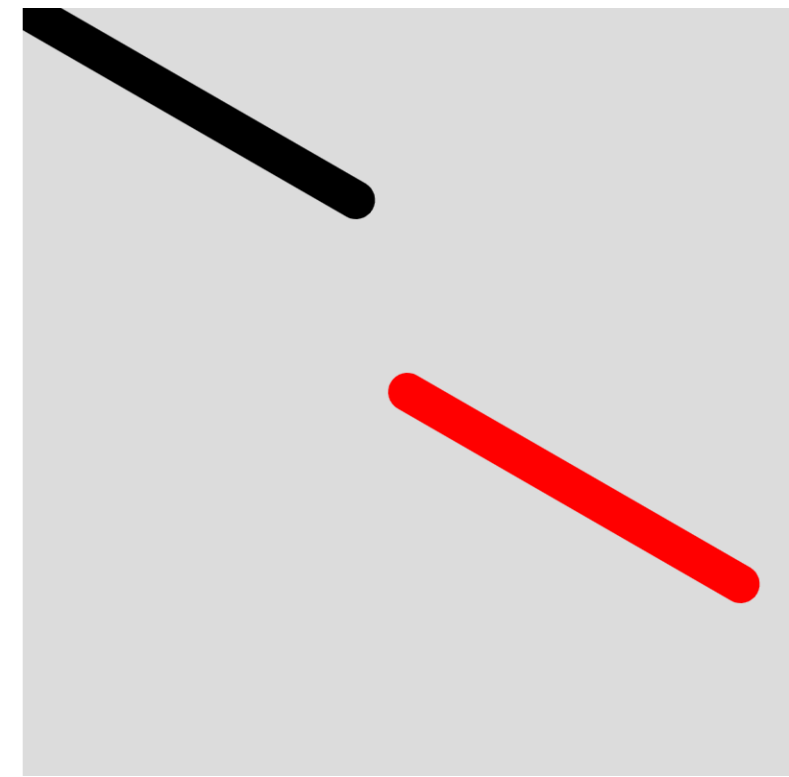
- This is a function we have created for CS106. It is not part of JavaScript p5.
- You must cut and paste it into your sketches when needed.
- The function accepts a radius in pixels and an angle in degrees, and returns a point (the point has an x and a y).

Our uWaterloo Polar function

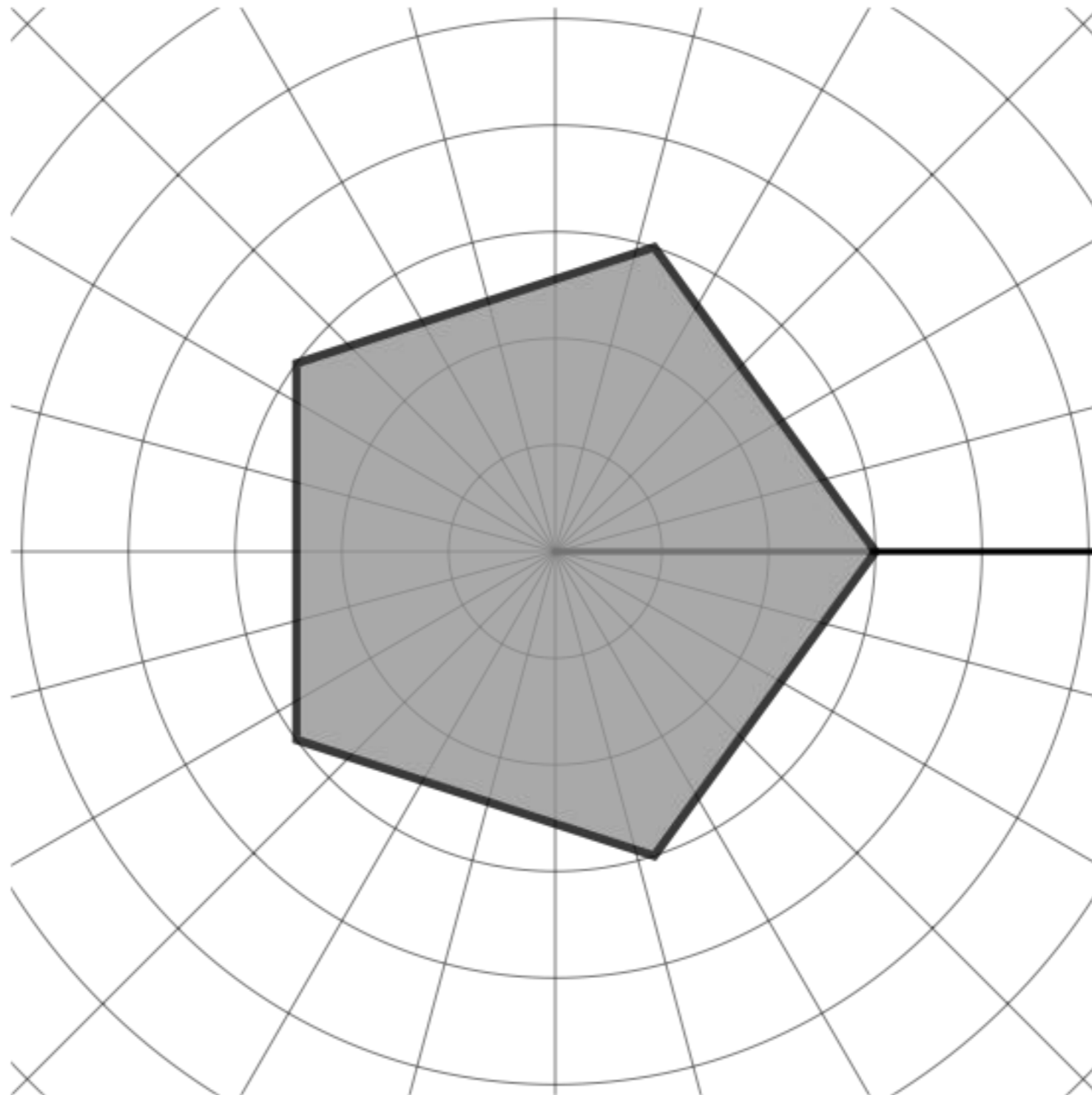
<https://openprocessing.org/sketch/1079187>

```
function setup() {  
  createCanvas(400, 400);  
  background(220);  
  strokeWeight(20);  
  
  let r = 200;  
  let ang = 30;  
  let coord = polar(r, ang);  
  
  line(0, 0, coord.x, coord.y) ;  
  stroke(255, 0, 0);  
  line( width/2, height/2, (width / 2) + coord.x, (height / 2) + coord.y);  
  
}
```

```
function polar(r, theta) {  
  let dt = radians(theta);  
  
  let x = r * cos(dt);  
  let y = r * sin(dt);  
  
  return createVector(x, y);  
  
}
```



Regular polygons



Regular Polygons (slide 1 of 2)

<https://openprocessing.org/sketch/1079214>

```
let num_sides = 4;

function setup() {
  createCanvas(500, 500);
}

function draw() {
  background(200);

  stroke(255);
  strokeWeight(2);
  fill(100);

  beginShape();
  for (let idx = 0; idx < num_sides; ++idx) {
    let ang = map(idx, 0, num_sides, 0, 360);
    let pt = polar(200, ang);
    vertex(width / 2 + pt.x, height / 2 + pt.y);
  }
  endShape(CLOSE);
}
```

Regular Polygons (slide 2 of 2)

<https://openprocessing.org/sketch/1079214>

```
function keyPressed() {  
  if (keyCode == UP_ARROW) {  
    ++num_sides;  
  } else {  
    num_sides = max(4, num_sides - 1);  
  }  
}
```

```
function polar(r, theta) {  
  let dt = radians(theta);  
  
  let x = r * cos(dt);  
  let y = r * sin(dt);  
  
  return createVector(x, y);  
}
```

Animating an ellipse (slide 1 of 2)

<https://openprocessing.org/sketch/1079224>

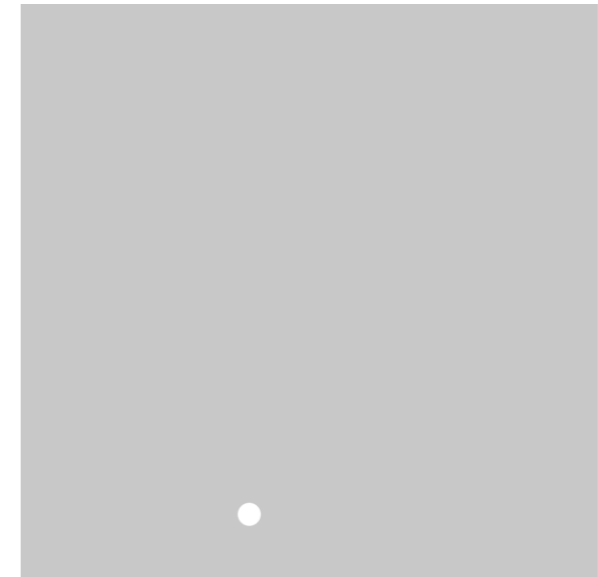
```
let ang = 0;

function setup() {
  createCanvas(500, 500);
}

function draw() {
  background(200);

  stroke(255);
  strokeWeight(2);
  fill(100);

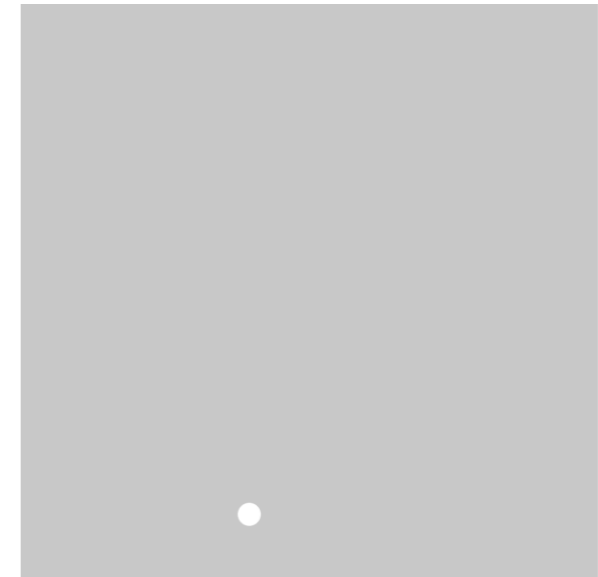
  ang = (ang + 1) % 360;
  let pt = polar(200, ang);
  strokeWeight(20);
  point(width / 2 + pt.x, height / 2 + pt.y);
}
```



Animating an ellipse (slide 2 of 2)

<https://openprocessing.org/sketch/1079224>

```
function polar(r, theta) {  
  let dt = radians(theta);  
  
  let x = r * cos(dt);  
  let y = r * sin(dt);  
  
  return createVector(x, y);  
}
```



Animating a Soccer Ball Image (slide 1 of 2)

<https://openprocessing.org/sketch/1079232>

```
let ang = 0;
let ball;

function preload() {
  ball = loadImage("ball.png");
}

function setup() {
  createCanvas(500, 500);
  imageMode(CENTER);
}

function draw() {
  background(200);

  stroke(255);
  strokeWeight(2);
  fill(100);

  ang = (ang + 1) % 360;
  let pt = polar(200, ang);
  image(ball, width / 2 + pt.x, height / 2 + pt.y);
}
```



Animating a Soccer Ball Image (slide 2 of 2)

<https://openprocessing.org/sketch/1079232>



```
function polar(r, theta) {  
  let dt = radians(theta);  
  
  let x = r * cos(dt);  
  let y = r * sin(dt);  
  
  return createVector(x, y);  
}
```