Drawing in Processing

The Basics

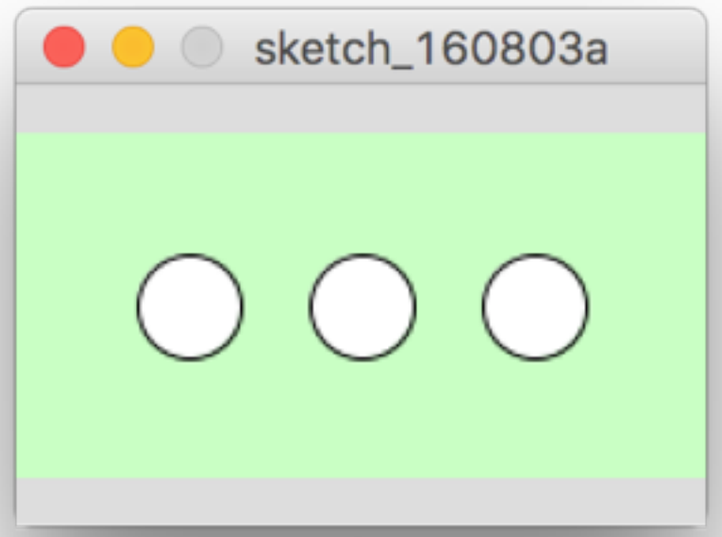
Static drawings

- Drawings in Processing can be static (not moving) or active (animated)
- In this lecture, we cover only static drawings
- Static drawings can be either:
  - With methods--must begin with a `void setup()` method
  - Without methods--just a list of statements and drawing commands
- The program should begin with a call to the `size(`***width***`, `***height***`)` method
  - The ***width*** and ***height*** must be given as literal integers, not as variables
  - They denote the size of the drawings, in pixels

A first static drawing

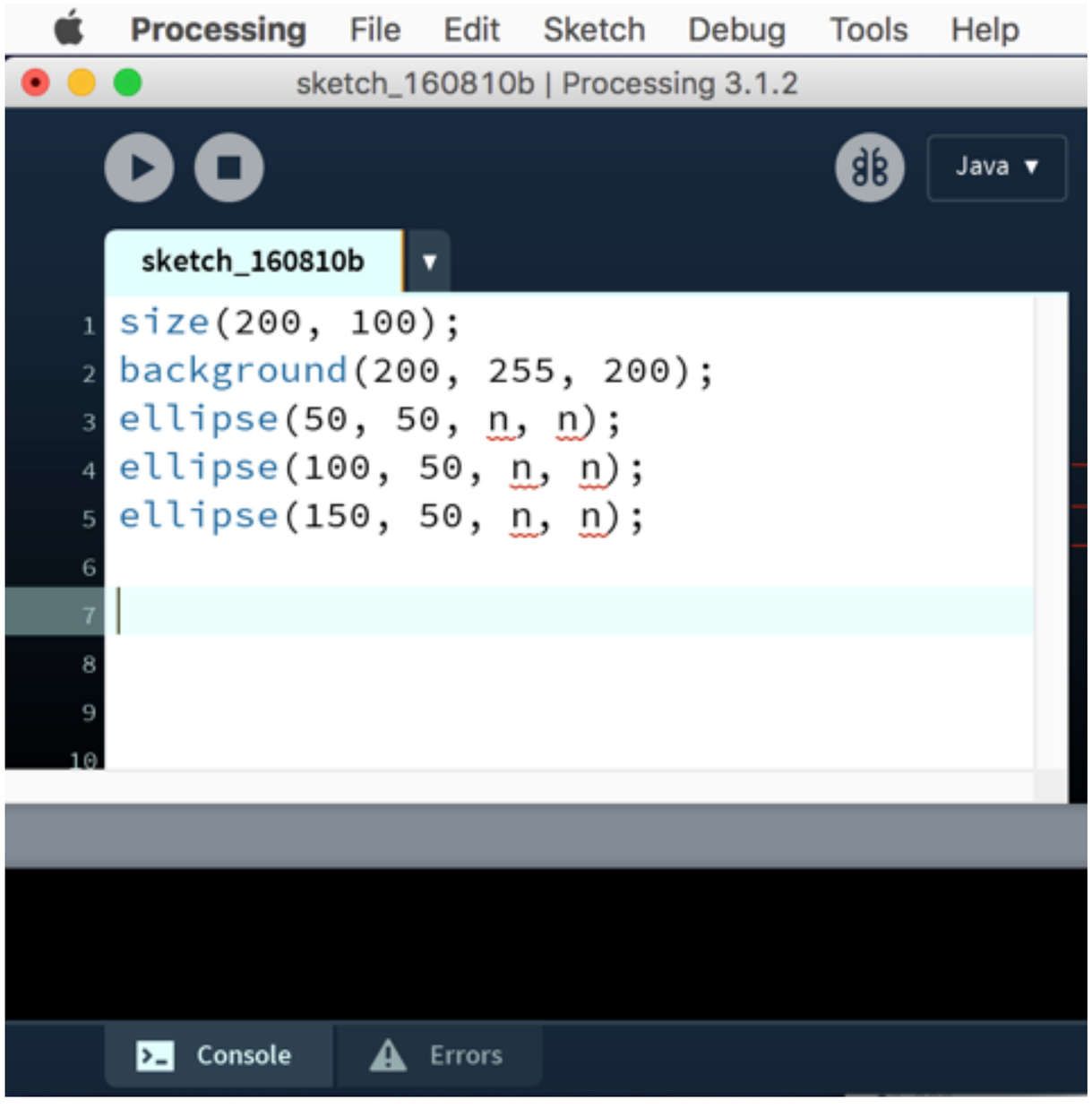
- Without methods:
- ```
  size(200, 100);
  int n = 30;
  background(200, 255, 200);
  ellipse(50, 50, n, n);
  ellipse(100, 50, n, n);
  ellipse(150, 50, n, n);
  ```

- With methods:
- ```
  void setup() {
      size(200, 100);
      background(200, 255, 200);
      doStuff(30);
  }

  void doStuff(int n) {
      ellipse(50, 50, n, n);
      ellipse(100, 50, n, n);
      ellipse(150, 50, n, n);
  }
  ```



The Processing IDE

- Processing comes with its own little Integrated Development Environment
- This is similar to IDLE, so you shouldn't have any trouble understanding it



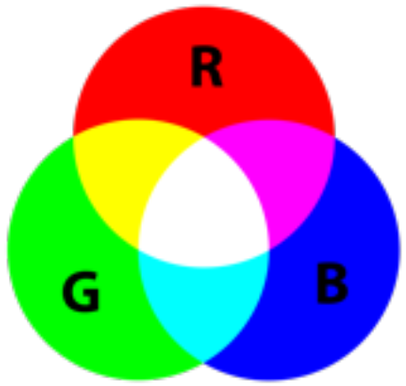
- This is a screenshot from a Mac; PC menus are similar
- Use the triangle to run the program, the square to stop the program

A note about numbers

- In Processing, as in Java, you can use an integer (such as `5`) anywhere that a floating point number (such as `5.0`) can be used
  - The reverse is not true; `int n = 5.0;` is illegal!
- In Processing, all numbers used in drawing (position on the screen, length and width, etc.) are `float`s
- Since the drawings in this presentation do not need to be very precise, I use integers a lot

Methods in the drawing

- `size(`***width***`,` ***height***`)` sets the size of the window
- `background(`***r***`,` ***g***`,` ***b***`)` sets the background color of the window
  - ***r***, ***g***, and ***b*** are integers in the range 0 to 255
  - `background(`***n***`)` sets a shade of gray: `0` = black, `255` = white

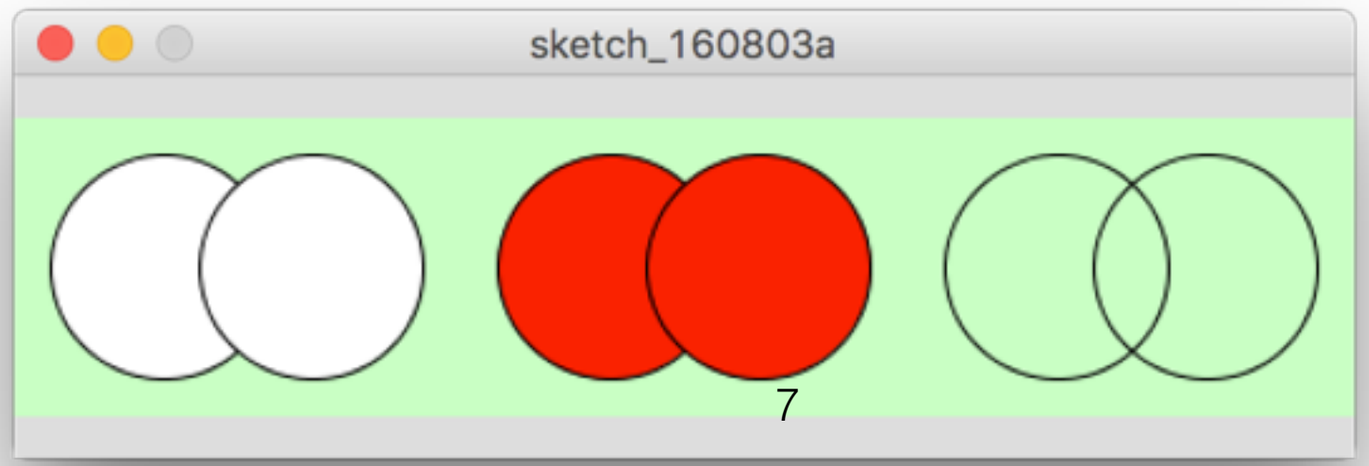


- `ellipse(`***x***`,` ***y***`,` ***width***`,` ***height***`)` draws an ellipse
  - x and y set the center of the ellipse
    - ***x*** is the distance from the left edge
    - ***y*** is the distance from the top
    - When ***width*** and ***height*** are equal, the result is a circle

`fill` and `noFill`

- `noFill();` says don't fill the inside of new figures
- `fill(g);` sets the internal shade of gray for new figures
- `fill(r, g, b);` sets the internal color for new figures

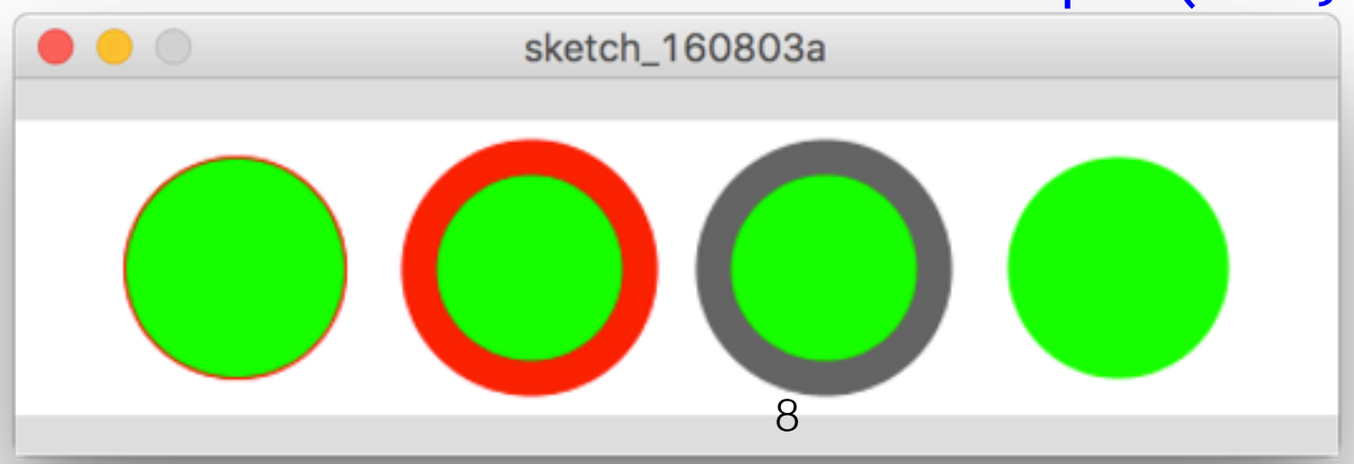
```
size(450, 100);
background(200, 255, 200);
int s = 75;

ellipse(50, 50, s, s);
ellipse(100, 50, s, s);

fill(255, 0, 0);
ellipse(200, 50, s, s);
ellipse(250, 50, s, s);

noFill();
ellipse(350, 50, s, s);
ellipse(400, 50, s, s);
```



stroke and noStroke

- `stroke(r, g, b);` sets the color for outlines of new figures
- `stroke(g);` sets the shade of gray for outlines of new figures
- `strokeWeight(w);` sets the thickness of new lines and outlines
- `noStroke();` says don't draw outlines of new figures

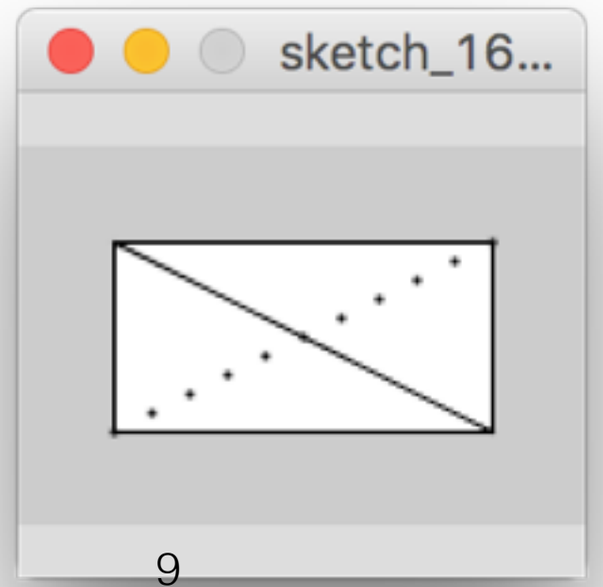
```
size(450, 100);
background(255);
fill(0, 255, 0);

stroke(255, 0, 0);
ellipse(75, 50, 75, 75);

strokeWeight(12);
ellipse(175, 50, 75, 75);

stroke(100);
ellipse(275, 50, 75, 75);

noStroke();
ellipse(375, 50, 75, 75);
```



rect, line, and point

- `rect(`*x*`, `*y*`, `*w*`, `*h*`);`
  draws a rectangle with (*x*, *y*) as the top left corner, *w* and *h* as the width and height
- `line(`*x1*`, `*y1*`, `*x2*`, `*y2*`);`
  draws a straight line from (*x1*, *y1*) to (*x2*, *y2*)
- `point(`*x*`, `*y*`);`
  draws a point *x* pixels from the left edge and *y* pixels from top

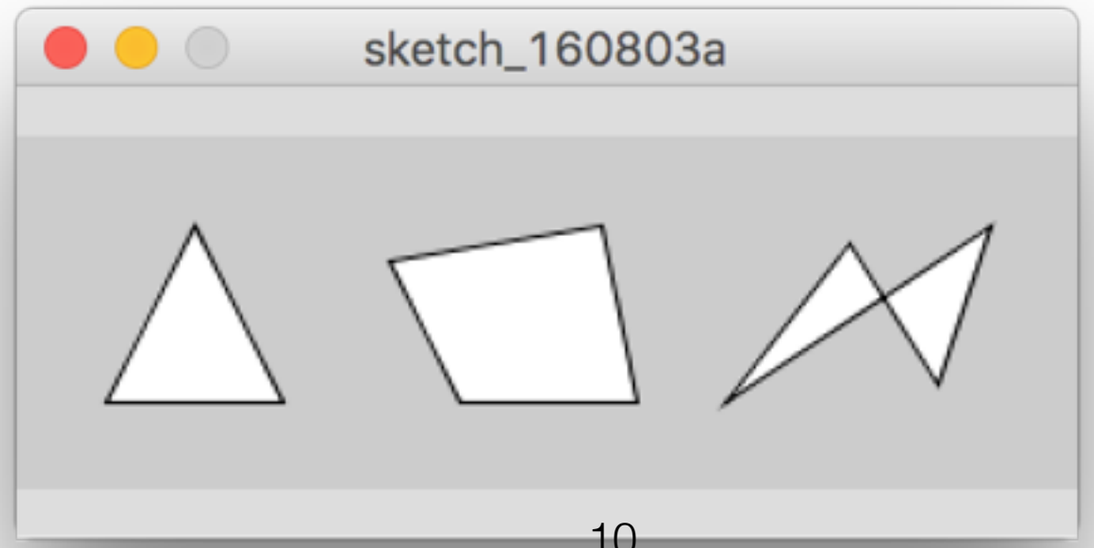
```
size(150, 100);

rect(25, 25, 100, 50);
line(25, 25, 125, 75);

strokeWeight(2);
int j = 75;
for (int i = 25; i <= 125; i += 10) {
  point(i, j);
  j -= 5;
}
```



triangle and quad

- `triangle(`*x1*`, `*y1*`, `*x2*`, `*y2*`, `*x3*`, `*y3*`);`
  draws a triangle connecting the three points
- `quad(`*x1*`, `*y1*`, `*x2*`, `*y2*`, `*x3*`, `*y3*`, `*x4*`, `*y4*`);`
  draws a quadrilateral connecting the four points

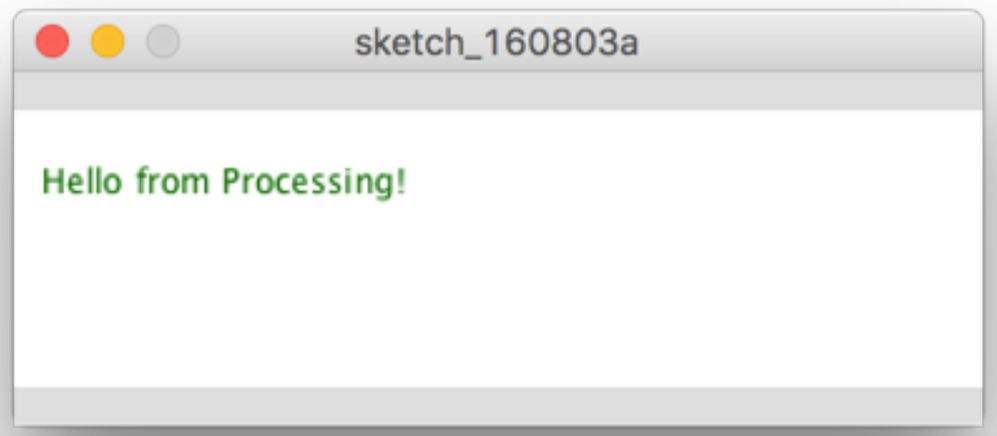
```
size(300, 100);

triangle(25,75, 50,25, 75,75);

quad(125,75, 175,75, 165,25, 105,35);

quad(200,75, 275,25, 260,70, 235,30);
```



text

- `text(s, x, y);`
  writes the string *s* at (*x*, *y*)
- The default font tends to be a bit small and dull

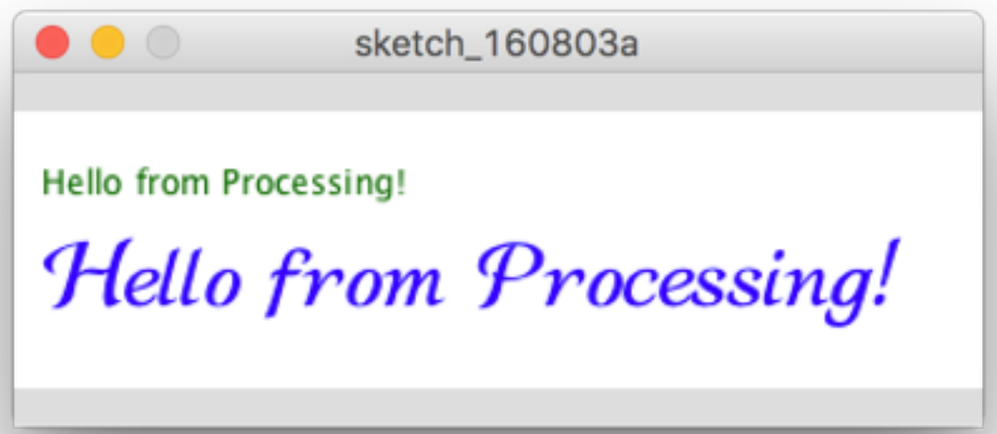
```
size(350, 100);
background(255);

fill(0, 100, 0);
text("Hello from Processing!", 10,
30);
```



Font

- PFont *font*; declares a variable that will hold a new Processing font
- createFont(*name*, *size*) creates a new Processing font
- textFont(*Pfont*) says to switch to using that font

```
PFont myFont;

size(350, 100);
background(255);

fill(0, 100, 0);
text("Hello from Processing!", 10, 30);
myFont = createFont("MMa Pascal", 32);
textFont(myFont);
fill(0, 0, 255);
text("Hello from Processing!", 10, 70);
```



Available fonts

- Not all fonts are available on all systems
- If you request a font that doesn't exist on your system, a default font will be used
  - This means that someone who runs your program on a different computer may not see exactly what you see
    - If you run my code from the previous slide, unless you happen to have `"MMa Pascal"`, you will see your default font
- The default font is sans serif; you can use the name `"Serif"` for a default serif font

arc

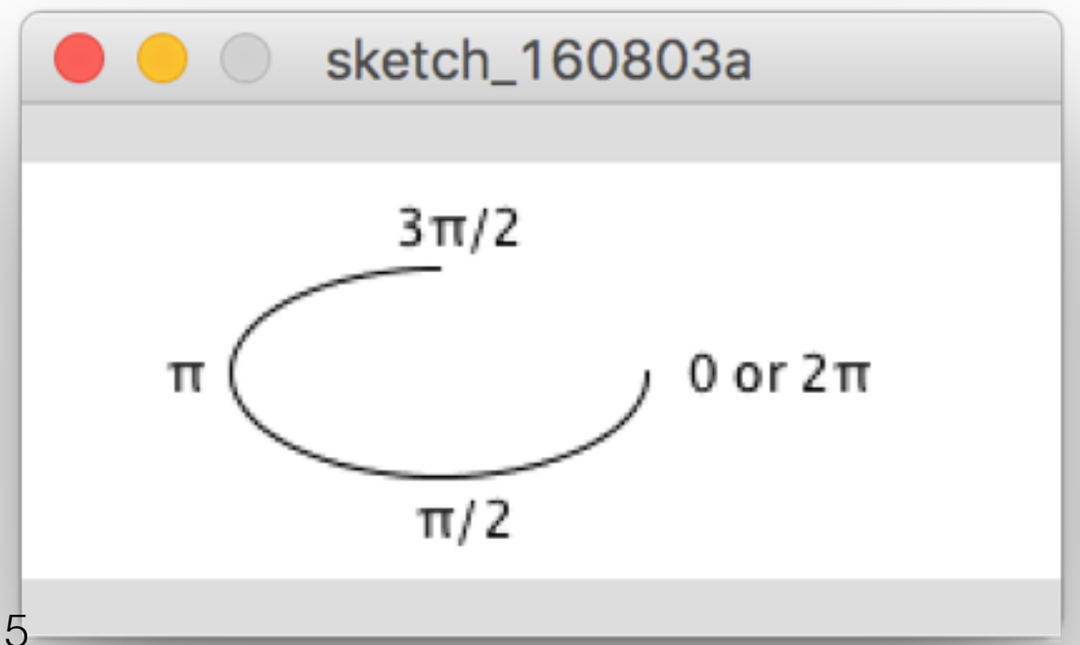
- `arc(`*`x`*`, `*`y`*`, `*`width`*`, `*`height`*`, `*`start`*`, `*`stop`*`, `*`mode`*`);`
  draws part of an ellipse whose center is at *x*, *y*
- *start* and *stop* are in radians
- Zero radians is the right side; going *start* to *stop* is going clockwise
- The mode is one of `OPEN` (default), `CHORD`, or `PIE`

```
size(350, 75);
background(255);

stroke(0);

arc(50, 25, 100, 50, 0, 2);
arc(150, 25, 100, 50, 0, 2, CHORD);
arc(250, 25, 100, 50, 0, 2, PIE);
```

Radians

- Radians are the “scientific” way of specifying angles
- 2π radians equals 360°
- In measuring arcs, zero is the extreme right edge
- The arc goes clockwise from start to stop
- Useful additional built-in constants are `PI`, `HALF_PI`, and `QUARTER_PI`

```
size(250, 100);
background(255);

stroke(0);
arc(100, 50, 100, 50, 0, PI + HALF_PI);

fill(0);
text("0 or 2π", 160, 55);
text("π", 35, 55);
text("π/2", 95, 90);
text("3π/2", 90, 20);
```



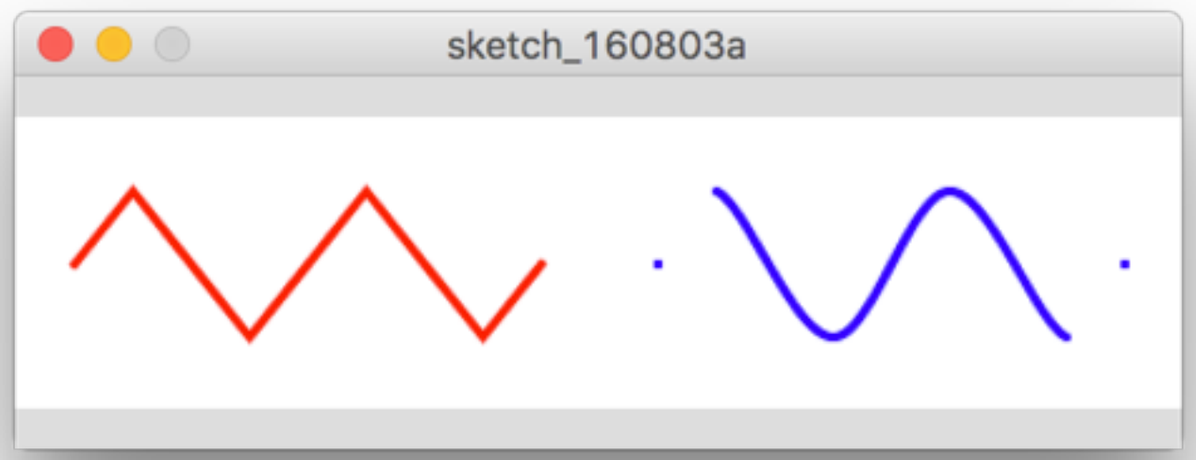
Shapes

- Multiple points can be grouped into a single curved or jagged line
  - `beginShape();`
    - Begins drawing a shape consisting of many straight lines or curves
    - Within the group you must have two or more calls to either `vertex` or to `curveVertex`, but not both kinds in the same shape
    - If the first and last vertices are the same, this draws a polygon
- `vertex(x, y);` defines end points of straight lines
- `curveVertex(x, y);` defines points along a curve
    - The first and last `curveVertex` points are *not* on the curve
    - To get a closed curve, duplicate the first and last vertices
- `endShape();`
  - Ends drawing the shape

Shapes

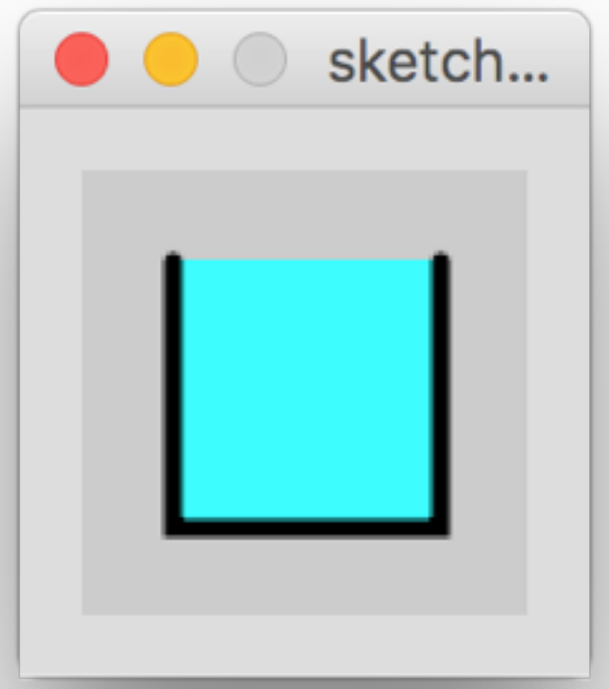
```
size(400, 100);
background(255.0);
strokeWeight(3);

stroke(255, 0, 0);
beginShape();
  vertex(20, 50);
  vertex(40, 25);
  vertex(80, 75);
  vertex(120, 25);
  vertex(160, 75);
  vertex(180, 50);
endShape();
```

```
stroke(0, 0, 255);
point(220, 50);
beginShape();
  curveVertex(220, 50);
  curveVertex(240, 25);
  curveVertex(280, 75);
  curveVertex(320, 25);
  curveVertex(360, 75);
  curveVertex(380, 50);
endShape();
point(380, 50);
```

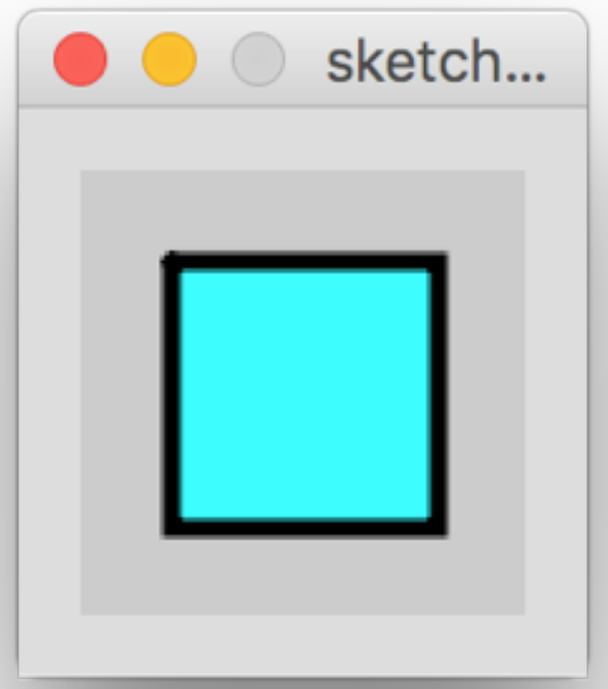


Closing shapes

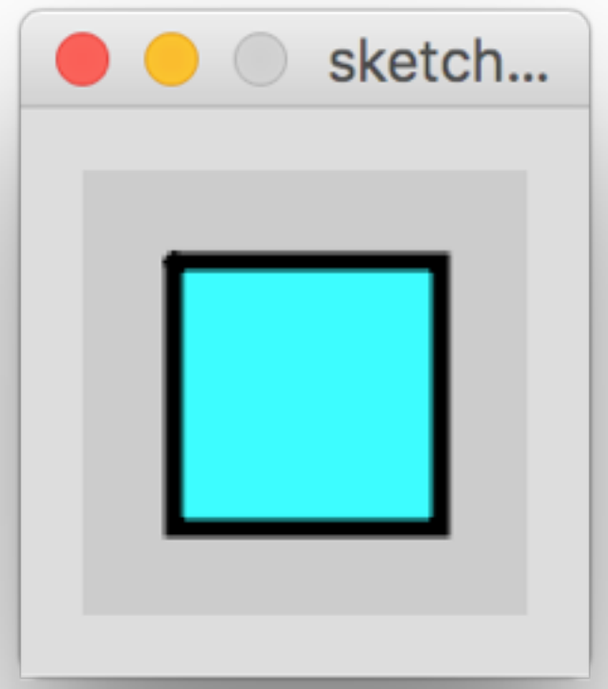
-
```
strokeWeight(4);
fill(0, 255, 255);
beginShape();
  vertex(20, 20);
  vertex(20, 80);
  vertex(80, 80);
  vertex(80, 20);
endShape();
```




```
strokeWeight(4);
fill(0, 255, 255);
beginShape();
  vertex(20, 20);
  vertex(20, 80);
  vertex(80, 80);
  vertex(80, 20);
  vertex(20, 20);
endShape();
```




```
strokeWeight(4);
fill(0, 255, 255);
beginShape();
  vertex(20, 20);
  vertex(20, 80);
  vertex(80, 80);
  vertex(80, 20);
endShape(CLOSE);
```



Bézier Curves

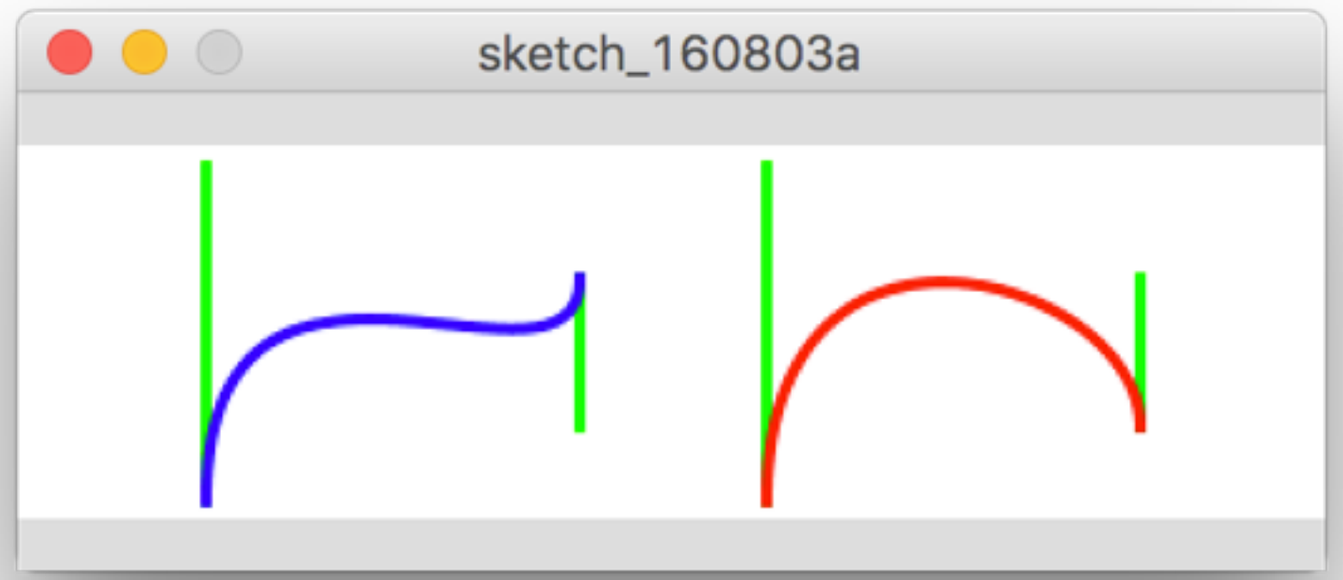
- The good news:
  - You can get curves of almost any shape you desire by using Bézier curves
  - If you get good at using Bézier curves, you can use them in almost every good drawing program
  - You aren't required to use them
- The bad news:
  - Bézier curves are tricky to master

- `bezier(`*`x1, y1, x2, y2, x3, y3, x4, y4`*`)`
  - The first and last points are anchor points
  - The middle two points are control points

- You can find tutorials on the web; here's one I like: http://learn.scannerlicker.net/2014/04/16/bezier-curves-and-type-design-a-tutorial/

Bézier Curves

```
size(350, 100);
background(255.0);
strokeWeight(3);

int a = 50, b = 95;
int c = 50, d = 5;
int e = 150, f = 75;
int g = 150, h = 35;
stroke(0, 255, 0); //green
line(a, b, c, d);
line(e, f, g, h);

stroke(0, 0, 255); // blue
bezier(a, b, c, d,
       e, f, g, h);
// Note: b > d and f > h

a += 150;
c += 150;
e += 150;
g += 150;
stroke(0, 255, 0); //green
line(a, b, c, d);
line(e, f, g, h);
stroke(255, 0, 0); //red
bezier(a, b, c, d, g, h, e, f);
// Note: e = g but now h > f
```



But wait…there's more

- This presentation covers only the very basics of 2d drawing in Processing
- https://processing.org/ is a great source for examples, tutorials, etc.
- I like to have this page open when programming: https://processing.org/reference/

The End