2D Shapes (using variables)

Creative Coding & Generative Art in Processing 2

Ira Greenberg, Dianna Xu, Deepak Kumar

Did you do this?

- Go the CS Computer Lab (Room 231 PSB)
- Log in
- Start the Processing application
  (Make sure it is Version 2.x)
- In a web browser, go to the Tutorials section of processing.org

  http://www.processing.org/tutorials/gettingstarted/

- Read the Getting Started tutorial (by Casey Reas & Ben Fry) and try out the two examples of simple Processing programs presented there
- If you'd like, install Processing 2.x on your own computer
- Do the Coordinate System and Shapes tutorial
- Read Ch. 2, pgs. 33-48 (for last class)
- Read Ch. 2, pgs. 48-63 (for this class)
- Start Assignment 1

Questions

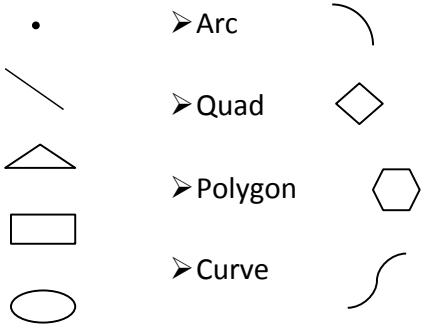
- Assignment1
- Reading for today
- Entry Survey (Please complete by next Tuesday)

Drawing Basics

- **Canvas** – **computer screen**
  `size(width, height);`
- **Drawing Tools** – **shape commands**
- **Colors** – **grayscale, RGB, or RGBA**
  `background(125);`

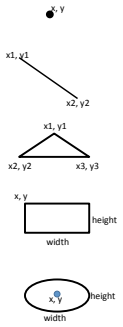
Drawing Tools - Basic Shapes

- Point
- Line
- Triangle
- Rectangle
- Ellipse
- Arc
- Quad
- Polygon
- Curve



Drawing Tools - Basic Shapes

- Point — point(x, y);
- Line — line(x1, y1, x2, y2);
- Triangle — triangle(x1, y1, x2, y2, x3, y3);
- Rectangle — rect(x, y, width, height);
- Ellipse — ellipse(x, y, width, height);

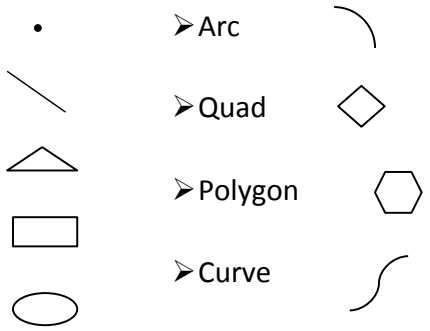


Drawing & Shape Attributes

- **Anti-aliasing**
  - smooth();
  - noSmooth();
- **Stroke**
  - noStroke();
  - strokeWeight(<pixel width>);
  - stroke(<stroke color>);
- **Fill**
  - noFill();
  - fill(<fill color>);

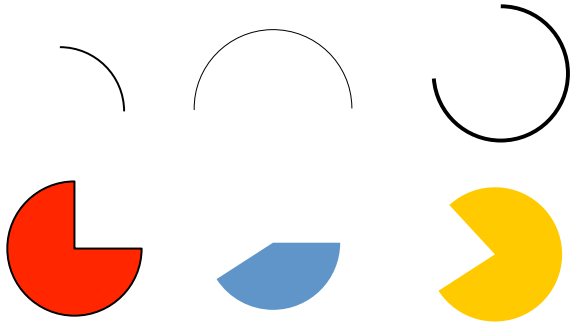
Drawing Tools - Basic Shapes

- Point
- Line
- Triangle
- Rectangle
- Ellipse
- Arc
- Quad
- Polygon
- Curve



Basic Shapes: Arcs

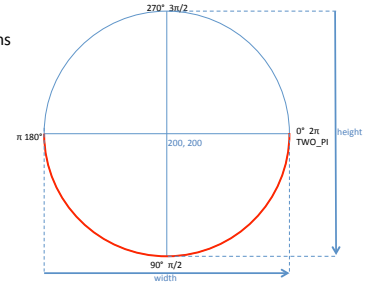
- What is an arc?



Basic Shapes: Arcs

```
arc(x, y, width, height, startAngle, endAngle);
```

- degrees *vs* radians

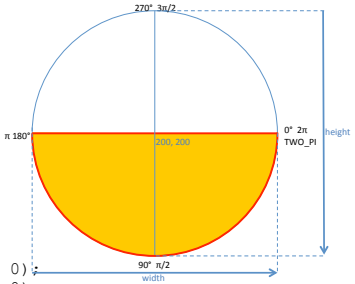



```
noFill();
stroke(255, 0, 0);
arc(200, 200, 150, 150, 0, PI);
```

Basic Shapes: Arcs

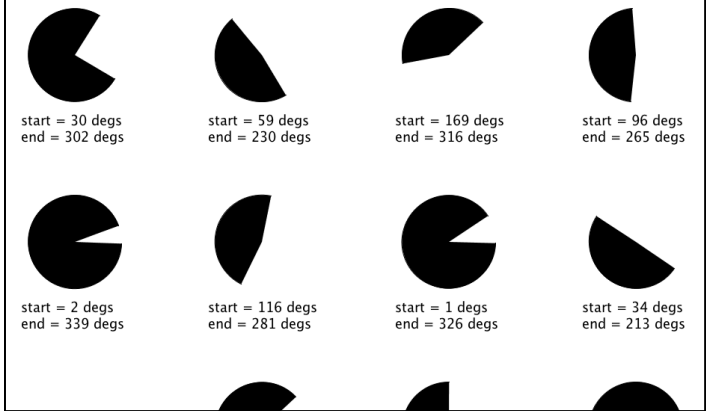
```
arc(x, y, width, height, startAngle, endAngle);
```

- degrees *vs* radians




```
fill(255, 255, 0);
stroke(255, 0, 0);
arc(200, 200, 150, 150, 0, PI);
```

Basic Shapes: Arcs



Basic Shapes: Quadrilaterals

```
quad(x1, y1, x2, y2, x3, y3, x4, y4);
```

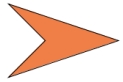
```
noStroke();
fill(12, 37, 80);
quad(100, 50, 150, 100, 100, 150, 50, 100);
```

```
fill(240, 127, 71);
quad(100, 50, 200, 50, 250, 100, 50, 100);
```

```
noStroke();
fill(163, 208, 193);
quad(100, 50, 150, 100, 100, 150, 250, 100);
```

Basic Shapes: Polygons

```
beginShape();
vertex(x1, y1);
…
vertex(xN, yN);
endShape(CLOSE);
```



```
fill(240, 127, 71);
beginShape();
  vertex(100, 50);
  vertex(150, 100);
  vertex(100, 150);
  vertex(250, 100);
endShape(CLOSE);
```

```
fill(240, 127, 71);
beginShape();
  vertex(100, 50);
  vertex(150, 100);
  vertex(100, 150);
  vertex(250, 100);
endShape();
```

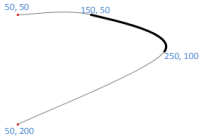
Basic Shapes: Curves

```
curve(cpx1, cpy1, x1, y1, x2, y2, cpx2, cpy2);
```

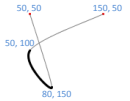
cpx1,cpy1- control point#1
x1, y1 - start of curve
x2, y2 - end of curve
cpx2,cpy2- control point#2

Draws a Catmull-Rom Spline between x1, y1 and x2, y2

Examples:




```
curve(50, 50, 150, 50, 250, 100, 50, 200);
```




```
curve(50, 50, 80, 150, 50, 100, 150, 50);
```

More Complex Curves

```
beginShape();
curveVertex(x1, y1);
…
curveVertex(xN, yN);
endShape(CLOSE);
```

```
curve(50, 50, 150, 50, 250, 100, 50, 200);
```

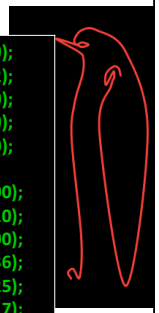
```
beginShape();
curveVertex(50, 50);
curveVertex(150, 50);
curveVertex(250, 100);
curveVertex(50, 200);
endShape();
```

Example: A Penguin

```
// penguin
size(400, 500);
smooth();

background(0);
stroke(245, 63, 55);
strokeWeight(3);
fill(0);
```

```
beginShape();
curveVertex(105, 400);
curveVertex(105, 400);
curveVertex(101, 392);
curveVertex(108, 387);
curveVertex(117, 398);
curveVertex(119, 342);
curveVertex(106, 210);
curveVertex(110, 160);
curveVertex(121, 120);
curveVertex(122, 99);
curveVertex(116, 90);

curveVertex(85, 72);
curveVertex(112, 80);
curveVertex(120, 83);
curveVertex(129, 80);
curveVertex(120, 77);
```

```
curveVertex(112, 80);
curveVertex(110, 72);
curveVertex(120, 60);
curveVertex(140, 60);
curveVertex(180, 90);

curveVertex(210, 200);
curveVertex(180, 410);
curveVertex(144, 200);
curveVertex(160, 136);
curveVertex(164, 125);
curveVertex(163, 117);
curveVertex(153, 135);
curveVertex(153, 120);
curveVertex(163, 110);
curveVertex(170, 112);
curveVertex(173, 122);
curveVertex(173, 122);

endShape();
```



Review: Drawing Basics

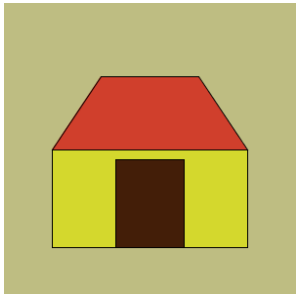
- **Canvas**
  `size(width, height)`
- **Drawing Tools**
  `point(x, y)`
  `line(x1, y1, x2, y2)`
  `triangle(x1, y1, x2, y2, x3, y3)`
  `quad(x1, y1, x2, y2, x3, y3, x4, y4)`
  `rect(x, y width, height)`
  `ellipse(x, y, width, height)`
  `arc(x, y, width, height, startAngle, endAngle)`
  `curve(cpx1, cpy1, x1, y1, x2, y2, cpx2, cpy2)`
  `beginShape()`
  `endShape(CLOSE)`
  `vertex(x, y)`
  `curveVertex(x, y)`
- **Colors**
  grayscale [0..255], **RGB** [0..255],[0..255],[0..255], **alpha** [0..255]
  `background(color)`
- **Drawing & Shape Attributes**
  `smooth(), noSmooth()`
  `stroke(color), noStroke(), strokeWeight(pixelWidth)`
  `fill(color), noFill()`

Simple Program Structure

```
// Create and set canvas
size(width, height);
smooth();
background(color);

// Draw something
…
// Draw something else
…
// etc.
```

Simple Program Structure

```
// Draw a barn
// Create and set canvas
size(300, 300);
smooth();
background(187, 193, 127);

// wall
fill(206, 224, 14);
rect(50, 150, 200, 100);

// Draw Door
fill(72, 26, 2);
rect(115, 160, 70, 90);

// Draw roof
fill(224, 14, 14);
quad(50, 150, 100, 75, 200, 75, 250, 150);
```



Variables: Naming Values

- **Values**
  42, 3.14159, 2013, "Hi, my name is Joe!", true, false, etc.
  - **Numbers**
    - **Integers**
      ```
      int meaningOfLife = 42;
      int year = 2013;
      ```
    - **Floating point numbers**
      ```
      float pi = 3.14159;
      ```
  - **Strings**
    ```
    String greeting = "Hi, my name is Joe!";
    ```
  - **Boolean**
    ```
    boolean keyPressed = true;
    ```

Variables: Naming Values

Variables have a Type

- **Values**
  42, 3.14159, 2013, "Hi, my name is Joe!", true, false, etc.
  - **Numbers**
    - **Integers**
      ```
      int meaningOfLife = 42;
      int year = 2013;
      ```
    - **Floating point numbers**
      ```
      float pi = 3.14159;
      ```
  - **Strings**
    ```
    String greeting = "Hi, my name is Joe!";
    ```
  - **Boolean**
    ```
    boolean keyPressed = true;
    ```

Variables: Naming Values

Variables have a Name

- **Values**
  42, 3.14159, 2013, "Hi, my name is Joe!", true, false, etc.
  - **Numbers**
    - **Integers**
      ```
      int meaningOfLife = 42;
      int year = 2013;
      ```
    - **Floating point numbers**
      ```
      float pi = 3.14159;
      ```
  - **Strings**
    ```
    String greeting = "Hi, my name is Joe!";
    ```
  - **Boolean**
    ```
    boolean keyPressed = true;
    ```

Variables: Naming Rules & Conventions

- Names begin with a letter, an underscore (_), or a dollar sign ($)

  Examples: `weight`, `_meaningOfLife`, `$value`
- Names may include numbers, but only after the initial character

  Examples: `value1`, `score5`, ~~`5bestFriends`~~
- No spaces are permitted in names

  Examples: ~~`value 1`~~, ~~`dollar sign`~~
- Processing Conventions
  - Names begin with a lowercase letter

    Example: `meaningOfLife`, `highestScore`
  - Constants are written in all caps

    Example: `DAYS_IN_WEEK`, `PI`

Variables: Declarations & Initialization

- Declaring variables

```
int meaningOfLife;
int year;
float pi;
String greeting;
boolean keyPressed;
```

- Initializing values in declarations

```
int meaningOfLife = 42;
int year = 2013;
float pi = 3.14159;
String greeting = "Hi, my name is Joe!";
boolean keyPressed = true;
```

The color type

- Processing has a type called color

```
color firebrick = color(178, 34, 34);
color chartreuse = color(127, 255, 0);
color fuchsia = color(255, 0, 255);
```

```
fill(firebrick);
rect(50, 100, 75, 125);
```

Expressions: Doing Arithmetic

- **Assignment statement**

  <variable> = <expression>;

  Examples:

```
meaningOfLife = 42;
area = length * height;
perc =statePop/totalPop*100.0;
```

- **Operators**

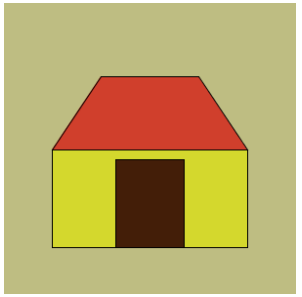
+	(addition)
-	(subtraction)
*	(multiplication)
/	(division)
%	(modulus)

Example:

```
mouth_x = ( (leftIris_x + irisDiam)/2 + eyeWidth )/4;
```

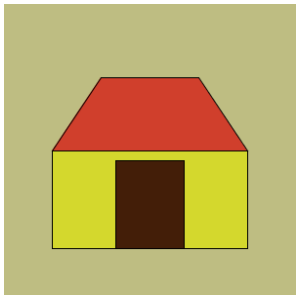
Using Variables

```
// Draw a barn
// Create and set canvas
size(300, 300);
smooth();
background(187, 193, 127);

// wall
fill(206, 224, 14);
rect(50, 150, 200, 100);

// Draw Door
fill(72, 26, 2);
rect(115, 160, 70, 90);

// Draw roof
fill(224, 14, 14);
quad(50, 150, 100, 75, 200, 75, 250, 150);
```



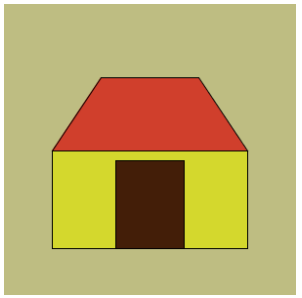
What variables will make this scalable?

```
// Draw a barn
// Create and set canvas
size(300, 300);
smooth();
background(187, 193, 127);

// wall
fill(206, 224, 14);
rect(50, 150, 200, 100);

// Draw Door
fill(72, 26, 2);
rect(115, 160, 70, 90);

// Draw roof
fill(224, 14, 14);
quad(50, 150, 100, 75, 200, 75, 250, 150);
```



What variables will make this movable?

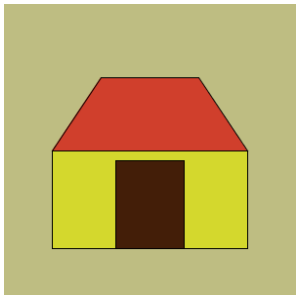
```
// Draw a barn
// Create and set canvas
size(300, 300);
smooth();
background(187, 193, 127);

// wall
fill(206, 224, 14);
rect(50, 150, 200, 100);

// Draw Door
fill(72, 26, 2);
rect(115, 160, 70, 90);

// Draw roof
fill(224, 14, 14);
quad(50, 150, 100, 75, 200, 75, 250, 150);
```



What variables will make this scalable and movable?

Pair up

1. Determine variables to use as references for position and size.
2. How many variables do you need for position?
3. How many for size?
4. Write a modified version of the program that uses the variables to specify each shape.



Homework

- Read Ch. 3 (pgs. 65-72)
- Read and do the **Color** tutorials on processing.org
- Review Processing commands:

  `size()`, `background()`, 2D shapes: `point()`, `line()`, `triangle()`, `rectangle()`, `quad()`, `ellipse()`. Attributes and modes: `stroke()`, `noStroke()`, `strokeWeight()`, `fill()`, `noFill()`, `rectMode()`, `ellipseMode()`.

  Color values (grayscale and RGB) and transparency.
- Review the concepts of an algorithm, pseudocode, syntax, and sequencing
- Complete Assignment 1.