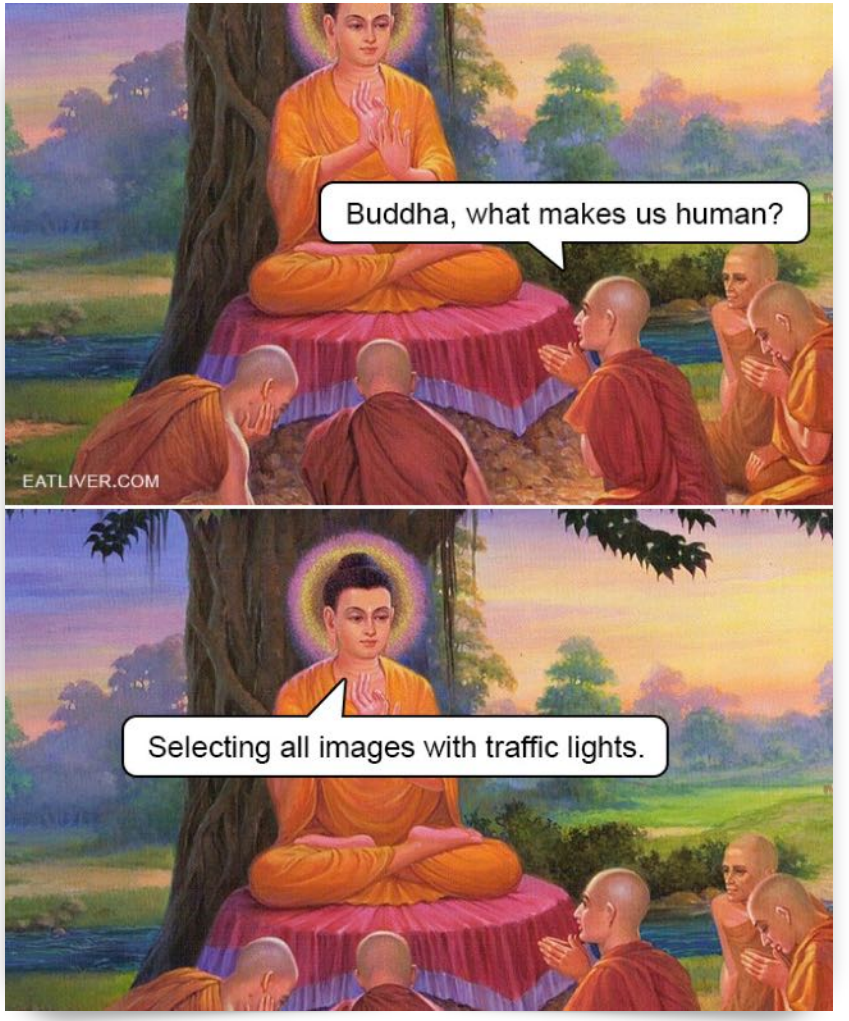
Introduction to Computer Programming

Dr Kafi Rahman, PhD
Email: kafi@truman.edu
Truman State University

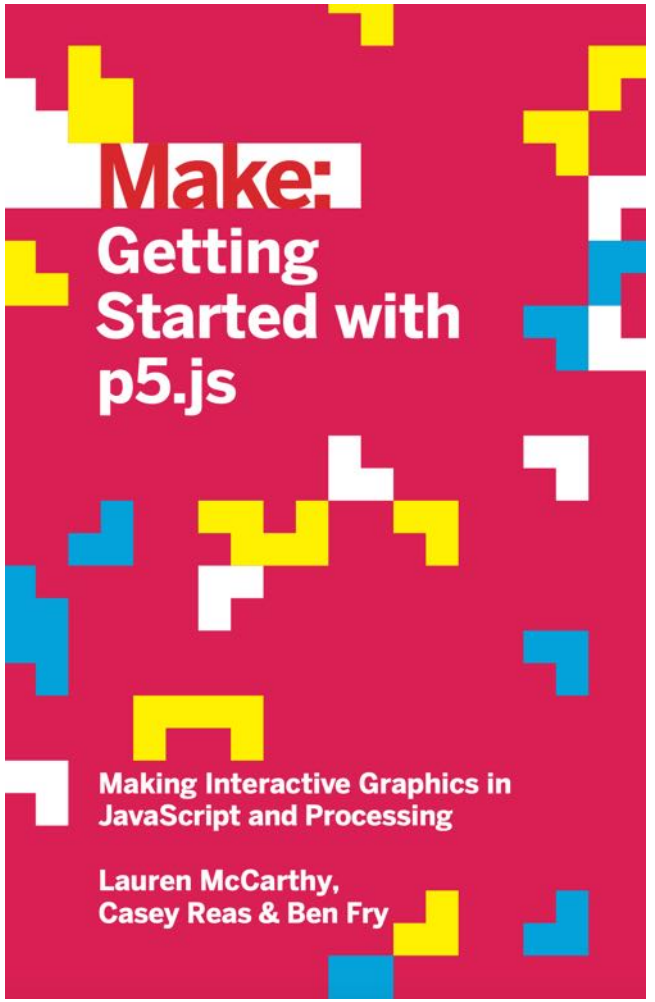
Game Rules

- We can use p5.js or any other game libraries
- You can use online game resources including
  - texture
  - sound
  - character texture



Useful Information

- Free book available on the website:
  - Getting started with p5.js



Software and Platforms

- Processing (p5.js)
  - The IDE is available on the website
  - link: https://editor.p5js.org
    - You should create an account to save your work
    - We shall demonstrate the steps to use the editor
- It is also possible to download the tools needed to code the language offline
  - We are not using this option for this course

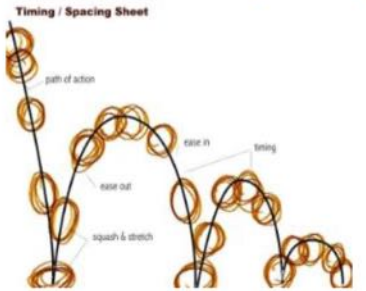
p5.js


Could you please go shopping and buy one jug of milk and, if they have avocados, get six.
They had avocados.
EATLIVER.COM

Animation (using Transformation)

- Geometric Transformations in 2D (note the z-coordinate is 0)Rapidly displaying sequence of images to create an illusion of movement
  - Flipbook
  - Keyframe animation: specify keyframes, computer interpolates (e.g., ball bouncing)

Flipbook

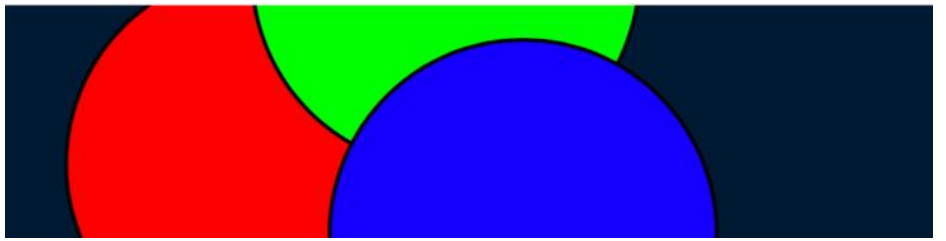


Keyframe Animation

Analyzing p5.js program

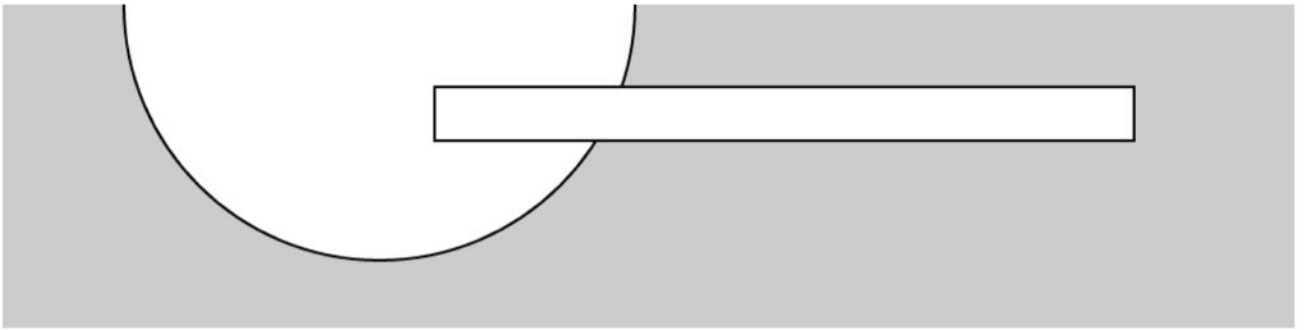
- Typical p5.js program

```
function setup() {
  createCanvas(480, 120);
  strokeWeight(2);
}

function draw() {
  // color of the window
  background(0, 26, 51);

  // first red ellipse
  fill(255, 0, 0);
  ellipse(132, 82, 200, 200);

  // drawing the green ellipse
  fill(0, 255, 0);
  ellipse(228, -16, 200, 200);

  // the blue ellipse parameters
  fill(0, 0, 255);
  ellipse(268, 118, 200, 200);
}
```



Control Your Drawing Order

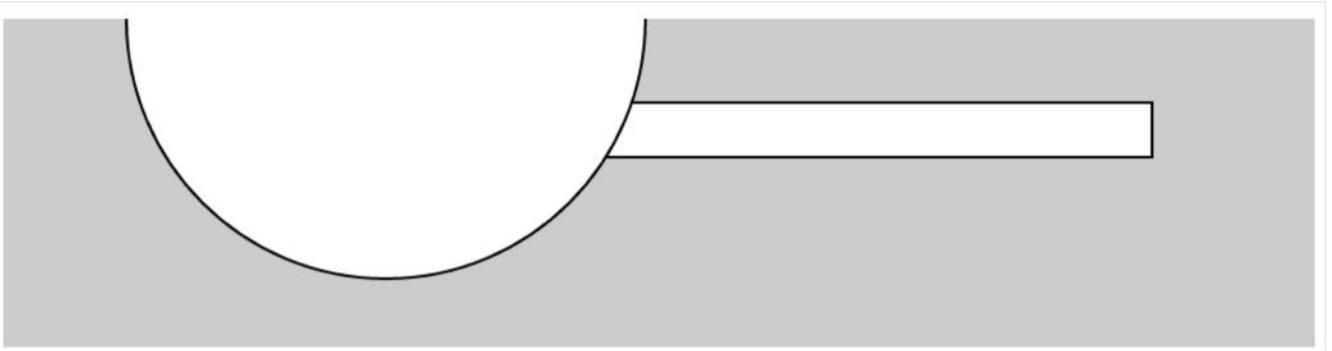
- If you want a shape to be drawn on top of all other shapes, it needs to follow the others in the code.



```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  ellipse(140, 0, 190, 190);
  // The rectangle draws on top of the ellipse
  // because it comes after in the code
  rect(160, 30, 260, 20);
}
```

Control Your Drawing Order

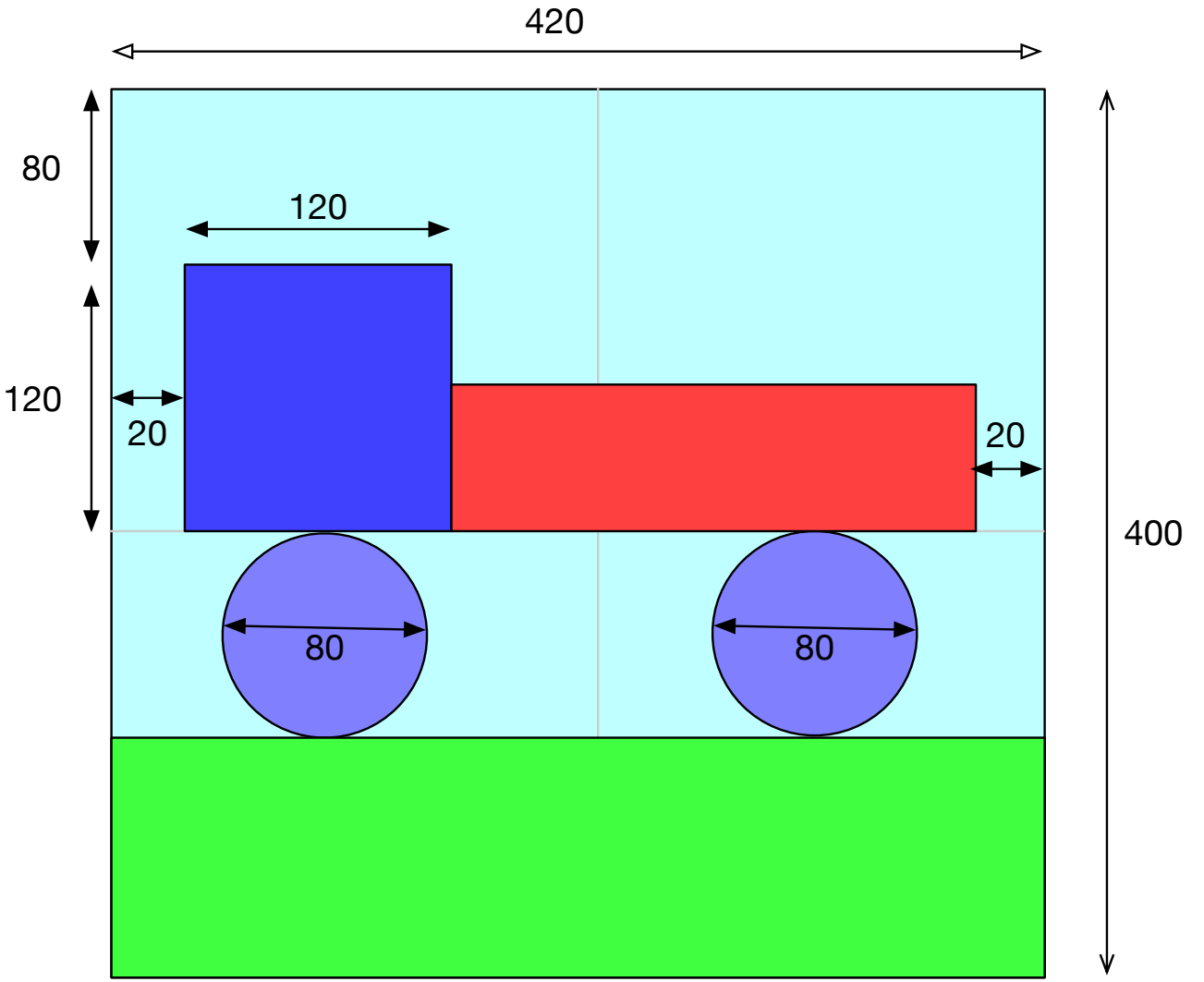
- Modify by reversing the order of rect() and ellipse() to see the circle on top of the rectangle:



```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  rect(160, 30, 260, 20);
  // The ellipse draws on top of the rectangle
  // because it comes after in the code
  ellipse(140, 0, 190, 190);
}
```

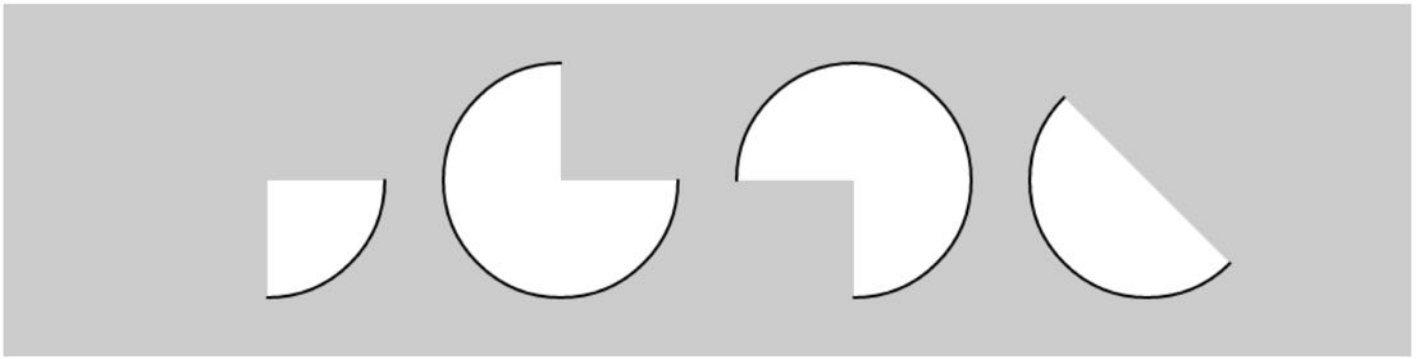
Let us draw together

- We have the following scene



Draw Part of an Ellipse

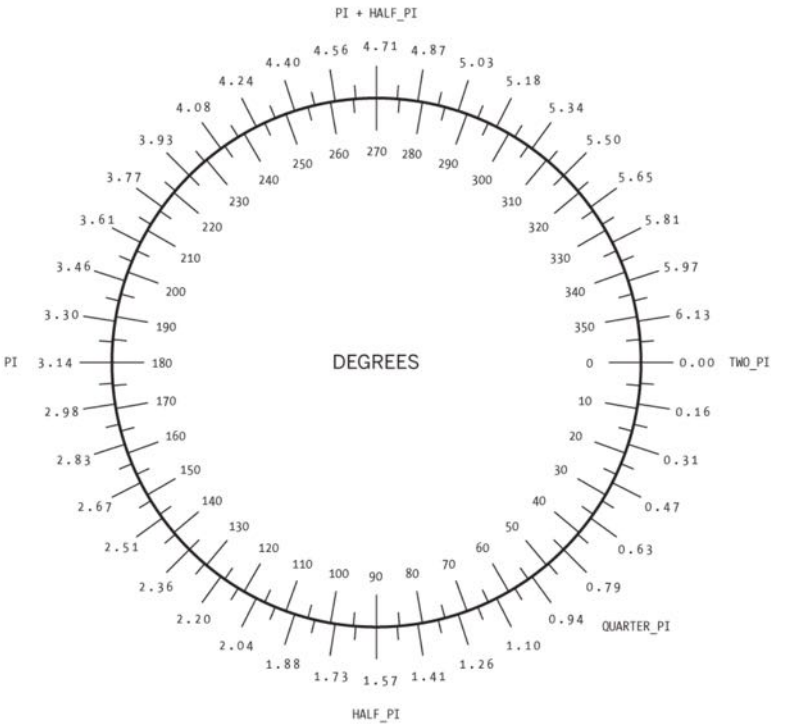
- The arc() function draws a piece of an ellipse:



```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  arc(90, 60, 80, 80, 0, HALF_PI);
  arc(190, 60, 80, 80, 0, PI+HALF_PI);
  arc(290, 60, 80, 80, PI, TWO_PI+HALF_PI);
  arc(390, 60, 80, 80, QUARTER_PI, PI+QUARTER_PI);
}
```

Draw Part of an Ellipse: the pi

- The first and second parameters set the location, while the third and fourth set the width and height.
- The fifth parameter sets the angle to start the arc and the sixth sets the angle to stop.
- The angles are set in radians, rather than degrees. Radians are angle measurements based on the value of pi (3.14159).



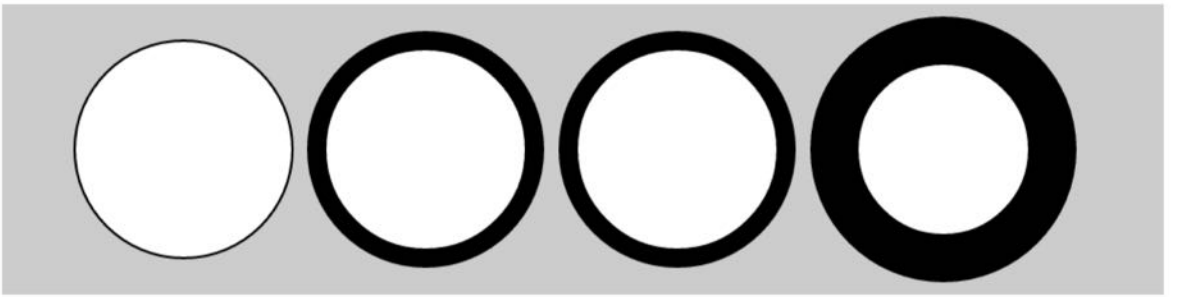
Draw Part of an Ellipse: angleMode function

- Convert your entire sketch to use degrees instead of radians using the angleMode() function.
  - This changes all functions that accept or return angles to use degrees or radians based on which parameter is passed in

```
function setup() {
  createCanvas(480, 120);
  angleMode(DEGREES);
}

function draw() {
  background(204);
  arc(90, 60, 80, 80, 0, 90);
  arc(190, 60, 80, 80, 0, 270);
  arc(290, 60, 80, 80, 180, 450);
  arc(390, 60, 80, 80, 45, 225);
}
```

Set Stroke Weight

- The default stroke weight is a single pixel, but this can be changed with the strokeWeight() function.
  - The single parameter to strokeWeight() sets the width of drawn lines:



```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  ellipse(75, 60, 90, 90);
  strokeWeight(8);  // Stroke weight to 8 pixels
  ellipse(175, 60, 90, 90);
  ellipse(279, 60, 90, 90);
  strokeWeight(20);  // Stroke weight to 20 pixels
  ellipse(389, 60, 90, 90);
}
```

Set Stroke Attributes

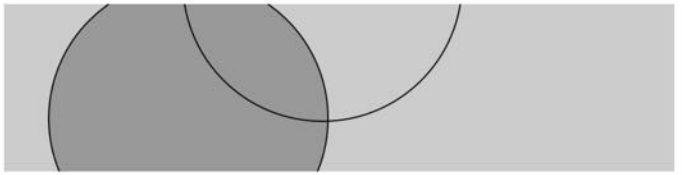
- The strokeJoin() function changes the way lines are joined (how the corners look), and the strokeCap() function changes how lines are drawn at their beginning and end:

```
function draw() {
  background(204);
  strokeJoin(ROUND);      // Round the stroke corners
  rect(40, 25, 70, 70);
  strokeJoin(BEVEL);      // Bevel the stroke corners
  rect(140, 25, 70, 70);
  strokeCap(SQUARE);      // Square the line endings
  line(270, 25, 340, 95);
  strokeCap(ROUND);       // Round the line endings
  line(350, 25, 420, 95);
}
```

Control Fill and Stroke

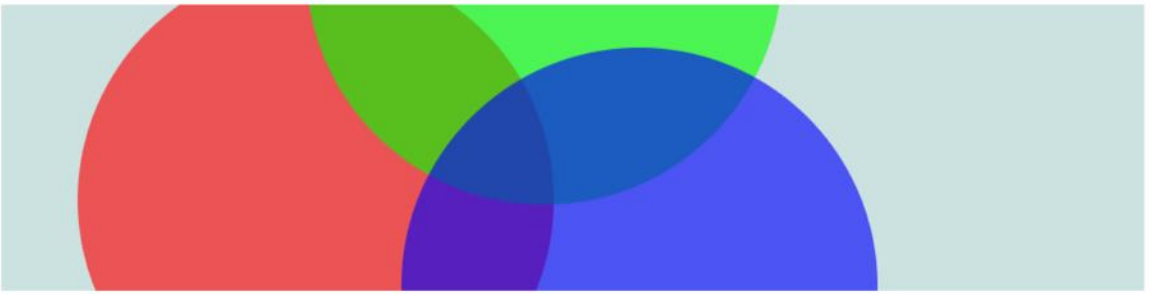
- You can use noStroke() to disable the stroke so that there's no outline, and you can disable the fill of a shape with noFill():

```
function draw() {
  background(204);
  stroke(0);
  fill(153);

  // Medium gray
  ellipse(132, 82, 200, 200);
  noFill();

  ellipse(228, -16, 200, 200);
  noStroke();

  // the following will not draw
  ellipse(268, 118, 200, 200);

}
```



Set Transparency

- By adding an optional fourth parameter to fill() or stroke(), we can control the transparency.
- This fourth parameter is known as the alpha value, and also uses the range 0 to 255 to set the amount of transparency.
  - The value 0 defines the color as entirely transparent (it won't display), the value 255 is entirely opaque



```
function setup() {
  createCanvas(480, 120);
  noStroke();
}

function draw() {
  background(204, 226, 225);    // Light blue color
  fill(255, 0, 0, 160);         // Red color
  ellipse(132, 82, 200, 200);   // Red circle
  fill(0, 255, 0, 160);         // Green color
  ellipse(228, -16, 200, 200);  // Green circle
  fill(0, 0, 255, 160);         // Blue color
  ellipse(268, 118, 200, 200);  // Blue circle
}
```

Assignment: Jack-o'-Lantern

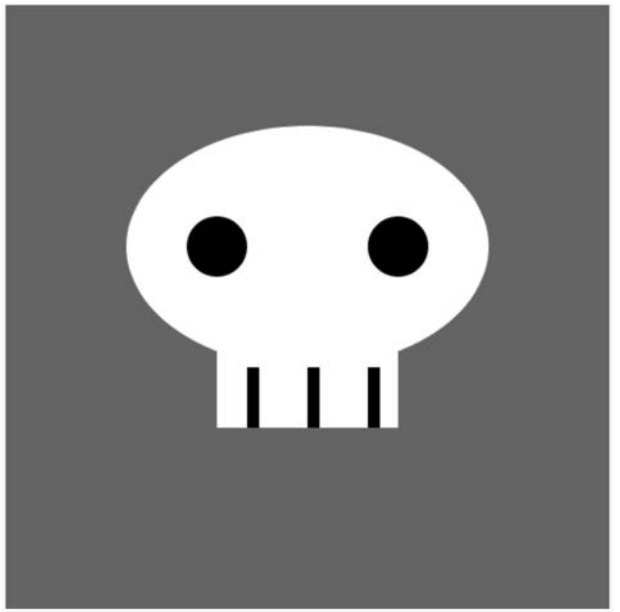
- By using the discussed processing commands, draw the following scene

Assignment: truck

- By using the discussed processing commands, draw the following scene

Assignment: Skull

- By using the discussed processing commands, draw the following scene

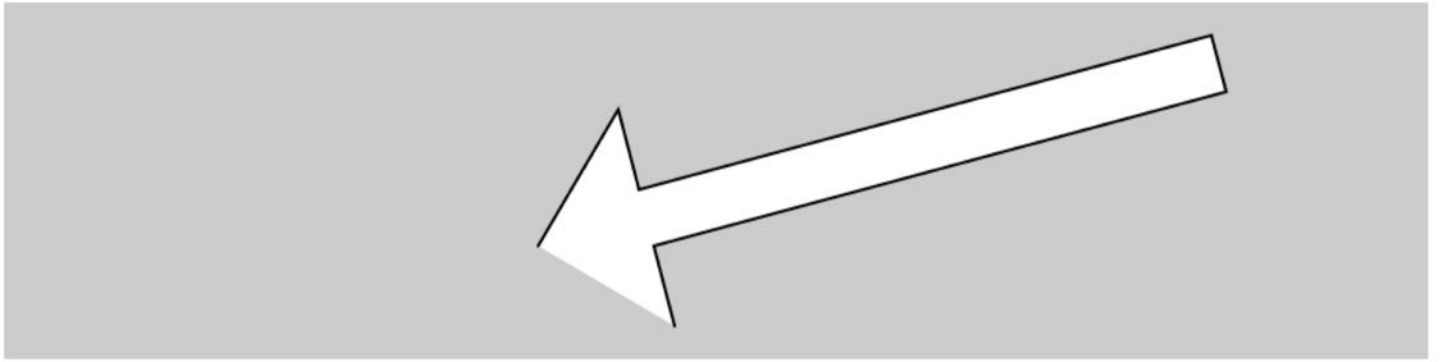


Custom Shapes

- We are not limited to using these basic geometric shapes—you can also define new shapes by connecting a series of points.

- The beginShape() function signals the start of a new shape.
- The vertex() function is used to define each pair of x and y coordinates for the shape.
- Finally, endShape() is called to signal that the shape is finished

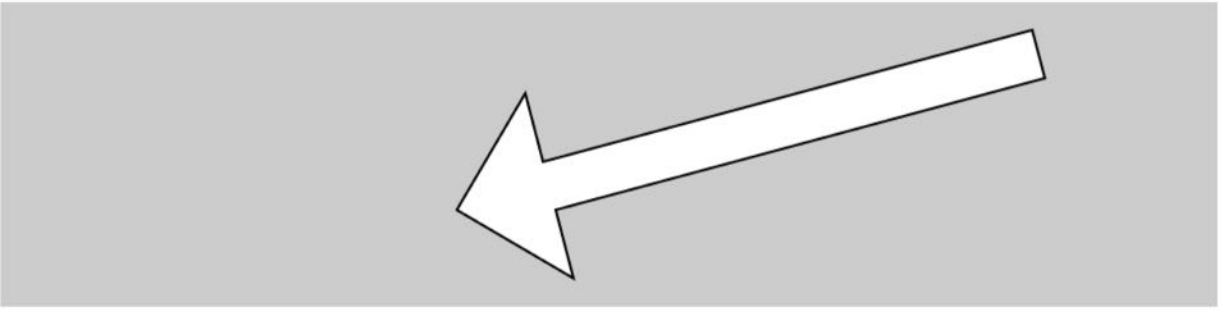
Draw an Arrow



```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  beginShape();
  vertex(180, 82);
  vertex(207, 36);
  vertex(214, 63);
  vertex(407, 11);
  vertex(412, 30);
  vertex(219, 82);
  vertex(226, 109);
  endShape();
}
```

Draw an Arrow: Close the Gap

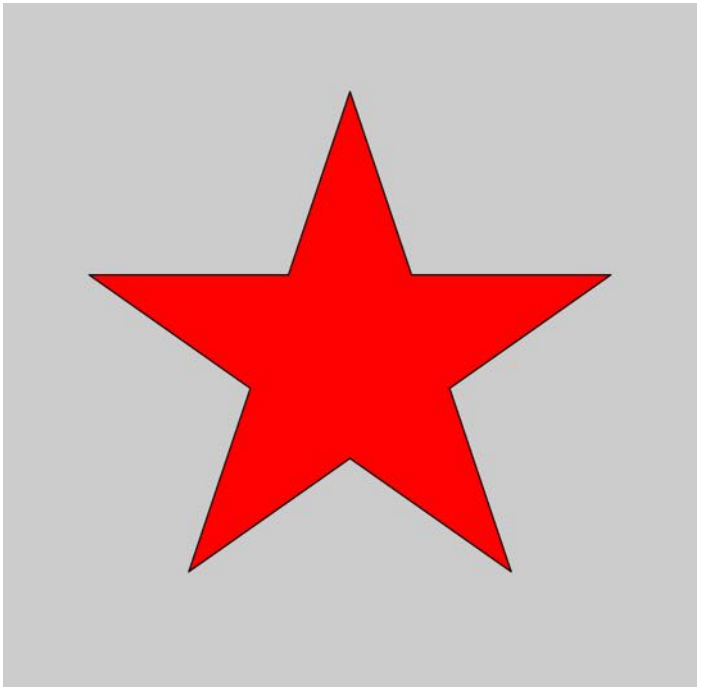
- To add the first and last points, add the word CLOSE as a parameter to endShape(), like this:



```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  beginShape();
  vertex(180, 82);
  vertex(207, 36);
  vertex(214, 63);
  vertex(407, 11);
  vertex(412, 30);
  vertex(219, 82);
  vertex(226, 109);
  endShape(CLOSE);
}
```

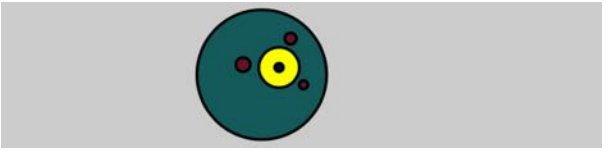
Practice: draw using vertex

- by using beginShape() and endShape() functions draw the following shape



Function: Grouping Statements

- We can group a set of statements and create a function
  - Enhances the organization of the code
  - In the following, we have created a function RobotHead and used it in the draw function



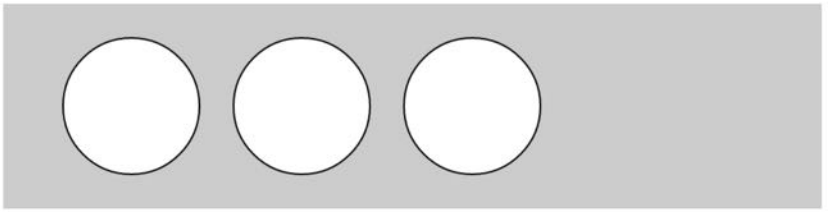
```
function RobotHead(){
  fill(20, 90, 90);
  ellipse(276, 155, 45, 45);
  fill(255, 255, 0); // eye
  ellipse(288, 150, 14, 14);
  fill(0, 20, 50); // eye small dot
  ellipse(288, 150, 3, 3);
  fill(102, 20, 40); // 3 dots
  ellipse(263, 148, 5, 5);
  ellipse(296, 130, 4, 4);
  ellipse(305, 162, 3, 3);
}

function draw() {
  background(204);
  // Head
  RobotHead();
}
```

```
function draw() {
  background(204);
  // Head
   fill(20, 90, 90);
  ellipse(276, 155, 45, 45);
  fill(255, 255, 0); // eye
  ellipse(288, 150, 14, 14);
  fill(0, 20, 50); // eye small dot
  ellipse(288, 150, 3, 3);
  fill(102, 20, 40); // 3 dots
  ellipse(263, 148, 5, 5);
  ellipse(296, 130, 4, 4);
  ellipse(305, 162, 3, 3);
}
```

Using Variables

Variable: Reuse the Same Values

- When you make the y coordinate and diameter for the three circles in this example into variables, the same values are used for each ellipse:

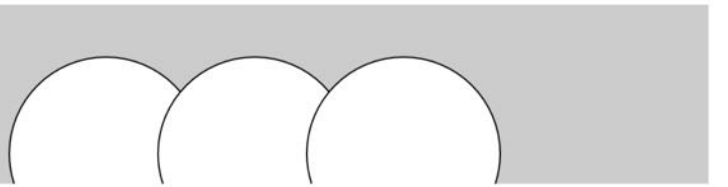


```
var y = 60;

var d = 80;

function setup() {
  createCanvas(420, 120);
}

function draw() {
  background(204);
  ellipse(75, y, d, d); // left
  ellipse(175, y, d, d); // middle
  ellipse(275, y, d, d); // right
}
```

Variable: Change Values

- Simply changing the y and d variables therefore alters all three ellipses:

```
var y = 100;
var d = 130;

function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  ellipse(75, y, d, d);   // Left
  ellipse(175, y, d, d);  // Middle
  ellipse(275, y, d, d);  // Right
}
```



Variable: Global Variables

- You can place your variables outside of setup() and draw().
- If you create a variable inside of setup(), you can't use it inside of draw(), so you need to place those variables somewhere else.
- Such variables are called global variables, because they can be used anywhere ("globally") in the program.

Variable: Displaying Variables

- Text can display the value of the variables
  - var x = 100;
  - text(x, 90, 50);
  - The above will display 100 at (90, 50) position

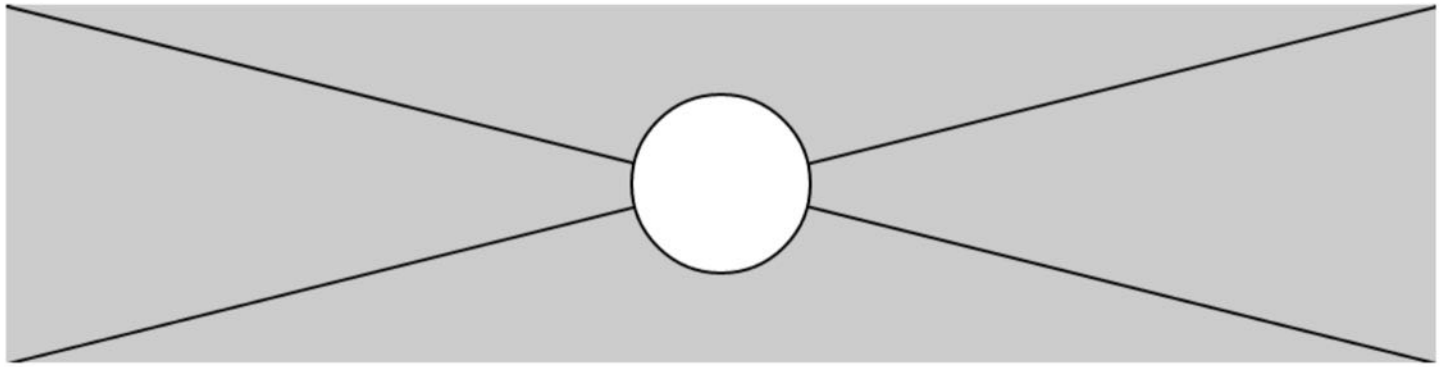
Variable: Displaying Variables

- Variables and text can be displayed together as well. For example,

```
var y = 60;
var d = 80;

function setup() {
  createCanvas(320, 120);
}

function draw() {
  background(204);

  text("y = " + y, 80, 40);
  text("d = " + d, 80, 60);
  text("Sum = " + (y+d), 80, 80);

}
```

Canvas Variables

- The width and height variables are available in the draw function

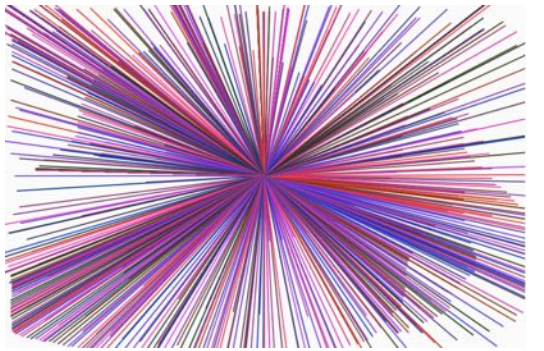


```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  line(0, 0, width, height);  // Line from (0,0) to (480, 120)
  line(width, 0, 0, height);  // Line from (480, 0) to (0, 120)
  ellipse(width/2, height/2, 60, 60);
}
```

Canvas Variables

- The width and height variables are available in the draw function
- Also, mouseX and mouseY variables are available that provides the current position of the mouse cursor on the canvas screen
- Given these variables, try to draw the following

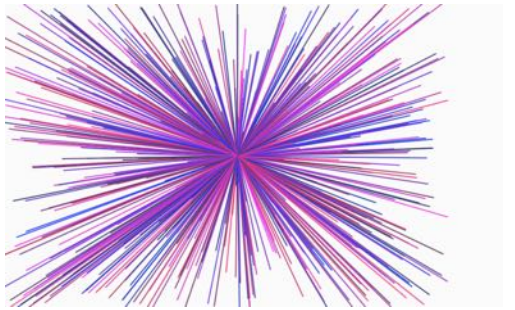
```
function draw() {
  //background(210);
  var x = 40, y = 30;

  stroke (random(255), 55, random(255));
  line(width/2, height/2, mouseX, mouseY);

}
```



Introducing: random function

- random function returns any value within the given range
  - for example, random(100) can return any random value between 0 and 100
- random function can also take two parameters
  - for example, random(10, 500) can return an random number between 10 and 500

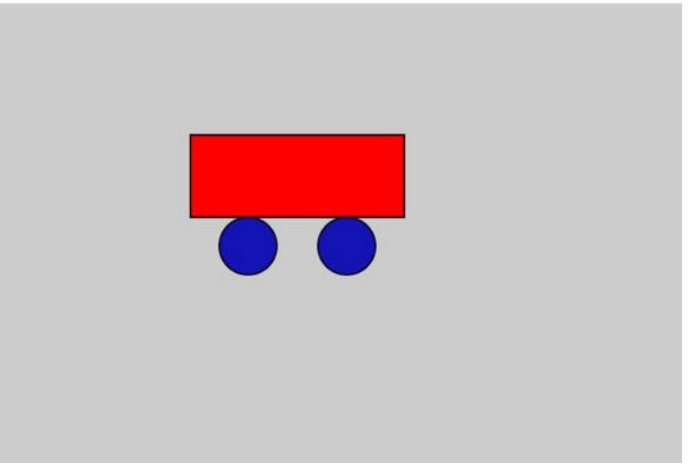
```
function draw() {
  //background(210);
  var x = 40, y = 30;

  stroke (random(10, 255), 55, random(90, 255));
  line(width/2, height/2, random(400), random(280));

}
```



Variables in drawing

- Location and size of a drawing can be specified by using variables
  - when the variables are changed, that will affect the drawing

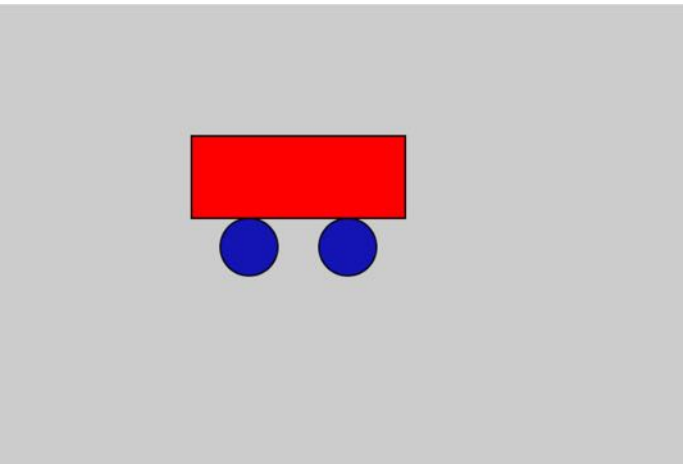
```
var y = 80, x = 120, r = 35;
var w = 130,h = 100;

function setup() {
  createCanvas(420, 280);
}

function draw() {
  background(204);

  // drawing a car
  fill(255, 0, 0);
  rect(x, y, w, h / 2);

  // left wheel
  fill(20, 20, 180);
  ellipse(x + r, y + h / 2 + r / 2, r, r);

  // right wheel
  fill(20, 20, 180);
  ellipse(x + w - r, y + h / 2 + r / 2, r, r);
}
```



Practice: Variables in drawing

- Now, add the x = x + 1; line in the program and see the change in action
  - when the car disappears, restart the program

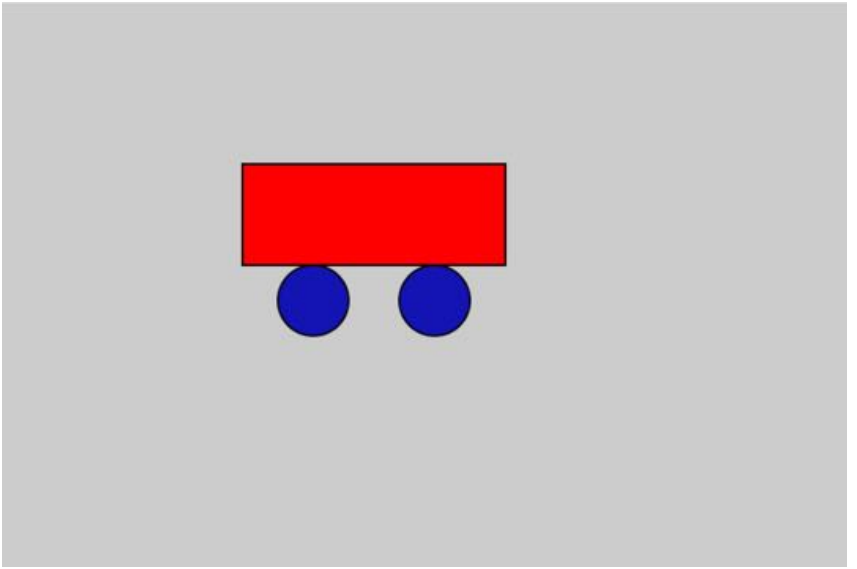
```
var y = 80, x = 120, r = 35;
var w = 130,h = 100;

function setup() {
  createCanvas(420, 280);
}

function draw() {
  background(100);
  x = x + 1; // change the location

  // drawing a car
  fill(255, 0, 0);
  rect(x, y, w, h / 2);

  // left wheel
  fill(20, 20, 180);
  ellipse(x + r, y + h / 2 + r / 2, r, r);

  // right wheel
  fill(20, 20, 180);
  ellipse(x + w - r, y + h / 2 + r / 2, r, r);
}
```



Practice: Variables in drawing

- Add, the mouseIsPressed condition to reset the position of the car
  - when the car disappears, click on the canvas screen

```
var y = 80, x = 120, r = 35;
var w = 130,h = 100;

function setup() {
  createCanvas(420, 280);
}

function draw() {
  background(100);
  if (mouseIsPressed) {
    x = 20;
    y = 80;
  }
  x = x + 1; // change the location

  // drawing a car
  fill(255, 0, 0);
  rect(x, y, w, h / 2);

  // left wheel
  fill(20, 20, 180);
  ellipse(x + r, y + h / 2 + r / 2, r, r);

  // right wheel
  fill(20, 20, 180);
  ellipse(x + w - r, y + h / 2 + r / 2, r, r);
}
```

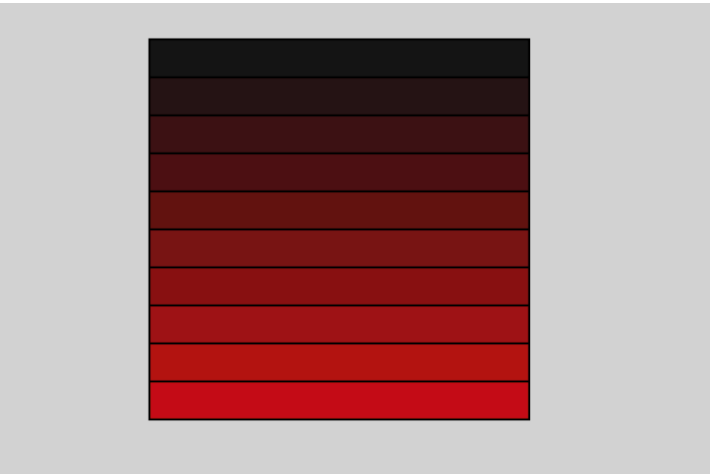


When you delete a block of code
that you thought was useless

Repetition: drawing

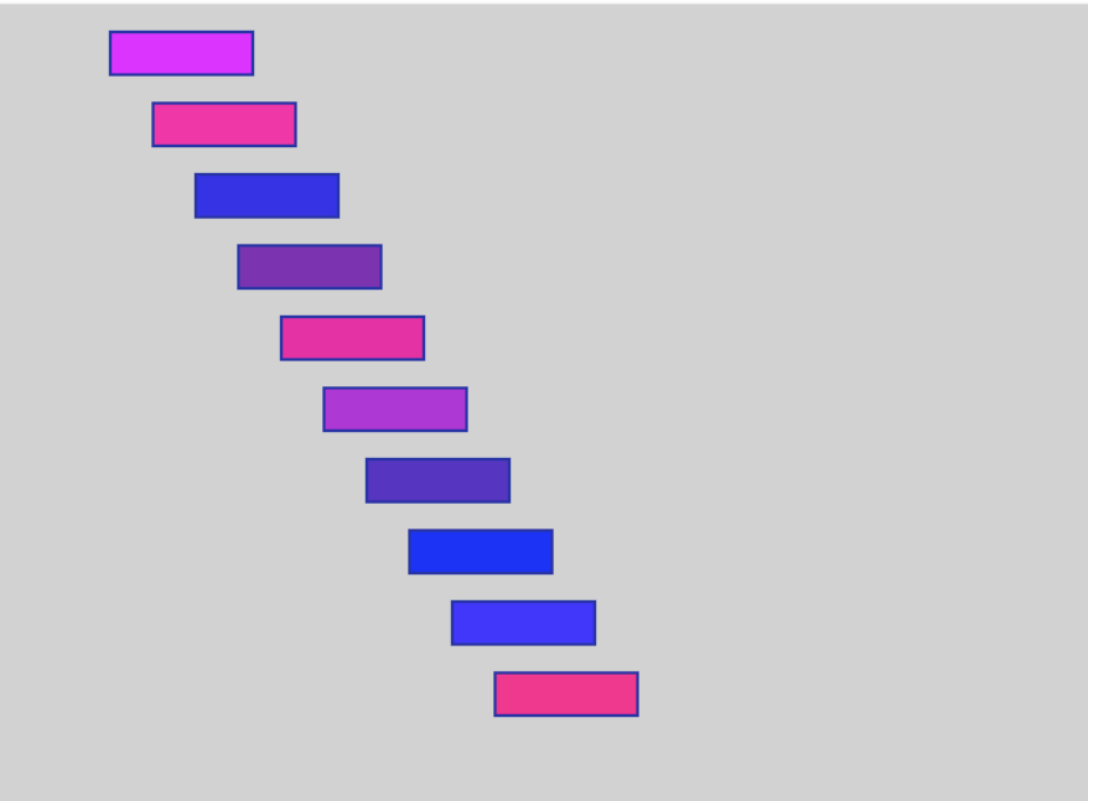
- By using for loop, we can repeat displaying the shapes
  - We accomplish that by changing the location of the shapes
  - In addition, we can change the color values as well

```
function draw() {
  background(210);

  var x = 80, y = 20;
  var value = 5;

  for(var i=1; i<=10; i++){

    fill(y, 20, 20);
    rect(x, y, 200, 20);

    y = y + 20;
  }
}
```



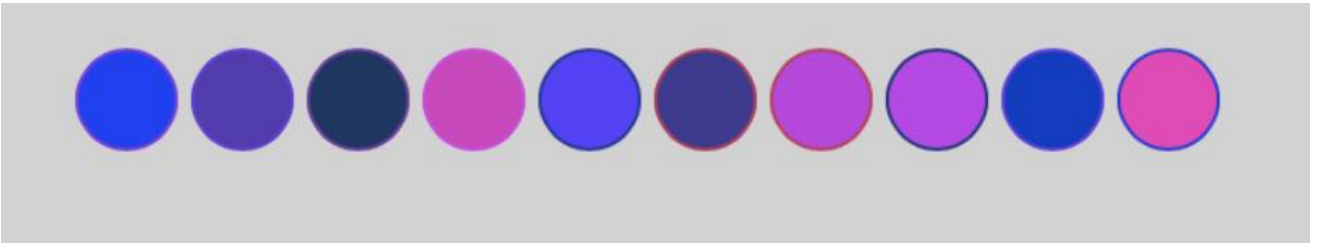
Assignment: drawing

- By using for loop, we can repeat displaying the shapes
  - draw the following by using a for loop (use random color for your shapes)



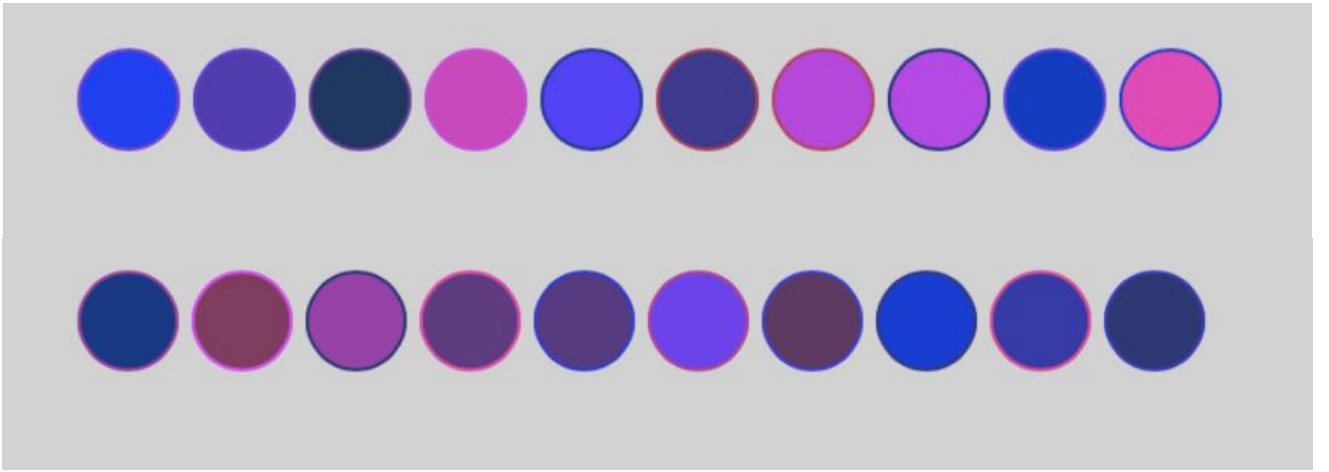
Assignment: drawing

- By using for loop, we can repeat displaying the shapes
  - By using processing commands, draw the following by using a for loop



Assignment: drawing

- By using for loop, we can repeat displaying the shapes
  - By using processing commands, draw the following by using a for loop



Event Handling

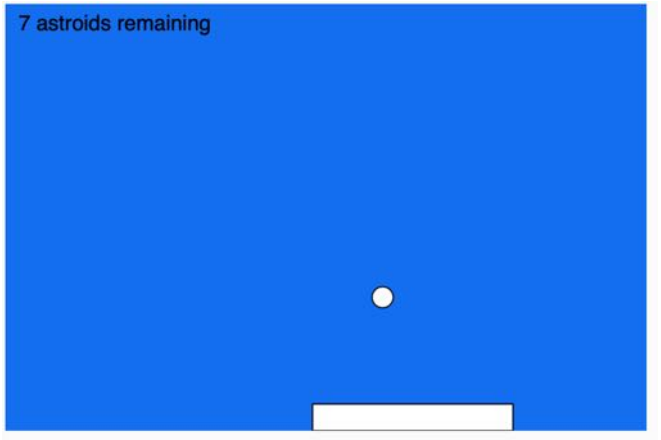
Move with Arrow Keys

- The following example shows how to check for the left or right arrow keys to move a rectangle:

```
var x = 215;

function setup() {
  createCanvas(480, 120);
}

function draw() {
  if (keyIsPressed) {
    if (keyCode == LEFT_ARROW) {
      x--;
    } else if (keyCode == RIGHT_ARROW) {
      x++;
    }
  }
  rect(x, 45, 50, 50);
}
```

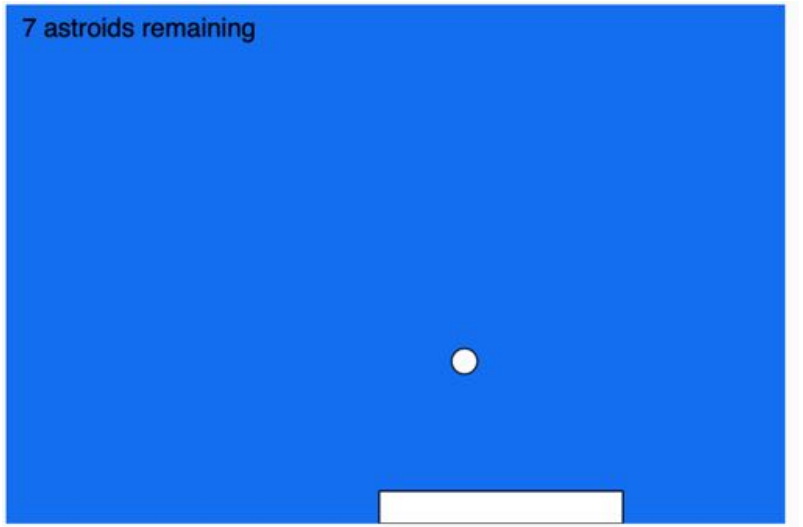
Assignment: Asteroid Defense

- Asteroids are falling from the sky
  - Protect the little rectangle from the asteroids

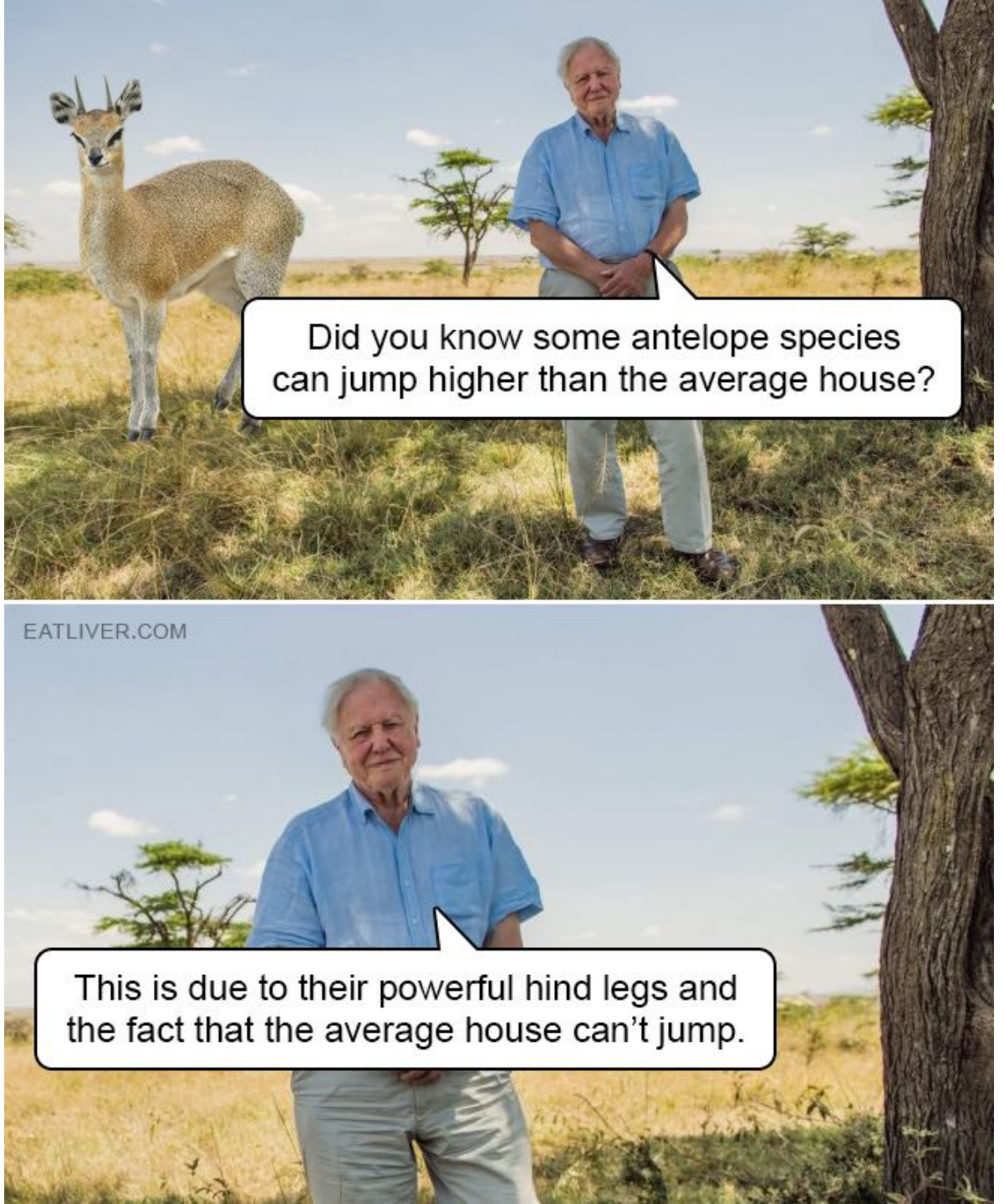


Assignment: Asteroid Defense

- In the setup function, generate random values for a circle
  - cy = -20 // so ellipse is at the top
- In the draw function do the following
  - Display the rectangle
  - Display the ellipse
  - The ellipse should move downwards
    - increase the value of cy variable by 5
- if the cy > height then generate random values for the circle again
- if the circle intersects with the rectangle then display "You Lost the Game"
- If the player can evade 8 asteroids then display "Congratulations! You Win the Game"



Transformation



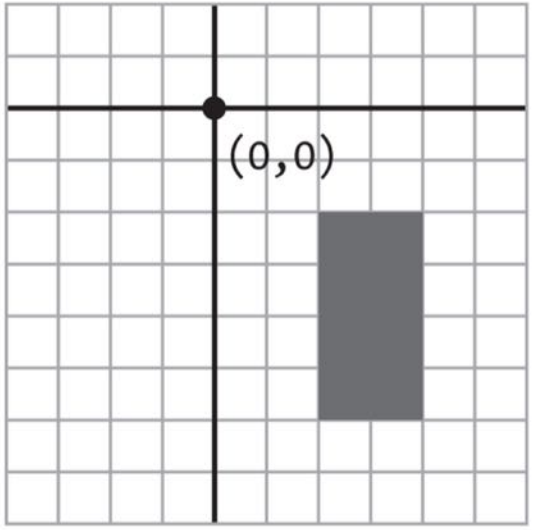
Transformation: Background

- An alternative technique for positioning and moving things on screen is to change the screen coordinate system.
- For example, you can move a shape 50 pixels to the right, or you can move the location of coordinate (0,0) 50 pixels to the right
  - the visual result on screen is the same.

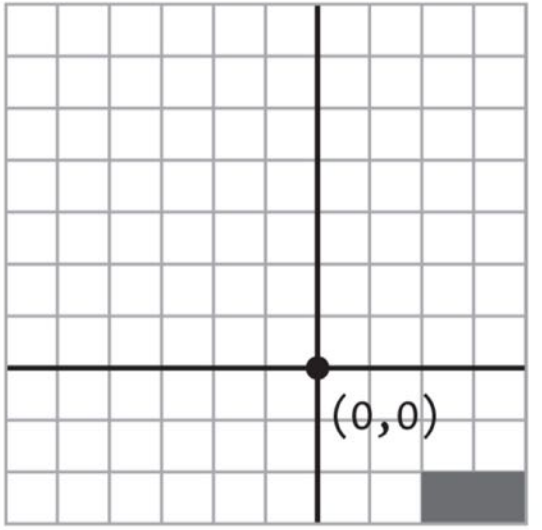
Translate

- translate() function is the most straightforward
  - this function can shift the coordinate system to the left, right, up, and down.

```
translate(40, 20);
rect(20, 20, 20, 40);
```




```
translate(60, 70);
rect(20, 20, 20, 40);
```



Translating Location

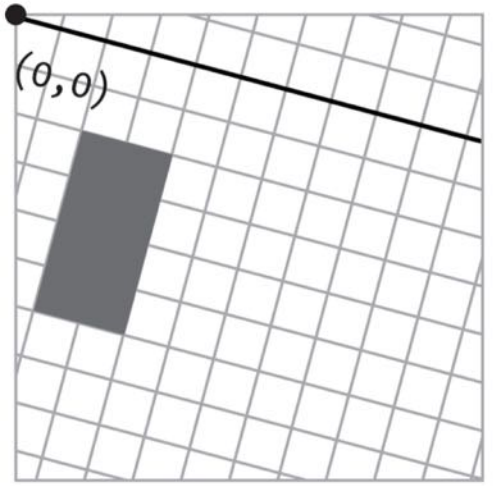
- In this example, notice that the rectangle is drawn at coordinate (0,0), but it is moved around on the canvas, because it is affected by translate():

```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  rect(0, 0, 30, 30);
}
```

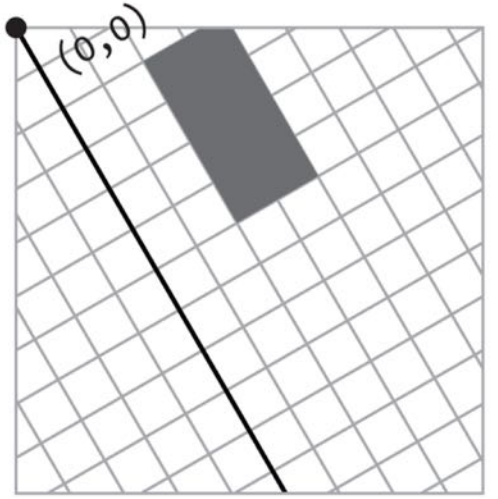
Rotate

- The rotate() function rotates the coordinate system. It has one parameter, which is the angle (in radians) to rotate.
- It always rotates relative to (0,0), known as rotating around the origin.

```
rotate(PI/12.0);
rect(20, 20, 20, 40);
```

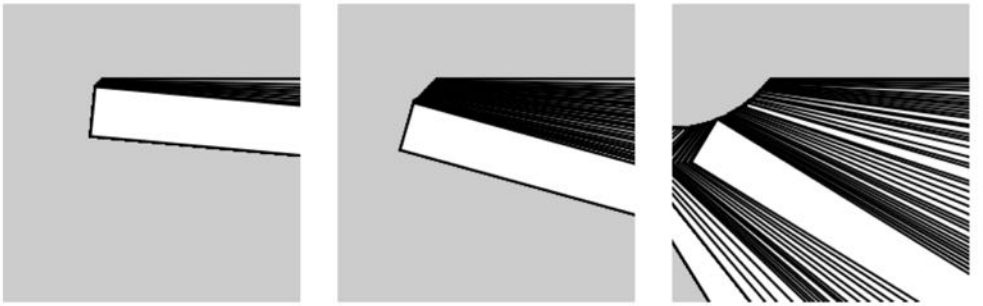



```
rotate(-PI/3);
rect(20, 20, 20, 40);
```



Corner Rotation

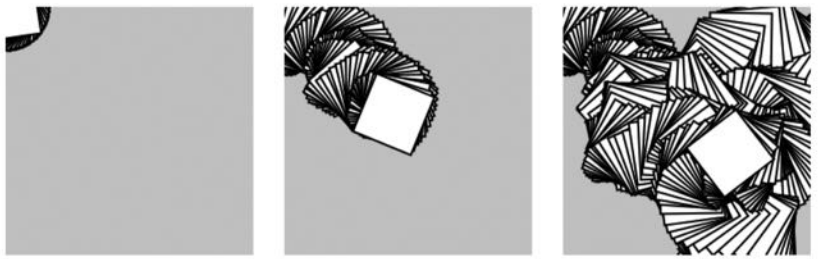
- To rotate a shape, first define the rotation angle with rotate(), then draw the shape.
- In this sketch, the parameter to rotate (mouseX / 100.0) will be between 0 and 1.2 to define the rotation angle because mouseX will be between 0 and 120, the width of the canvas as defined in createCanvas():



```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  rotate(mouseX / 100.0);
  rect(40, 30, 160, 20);
}
```

Translation, Then Rotation

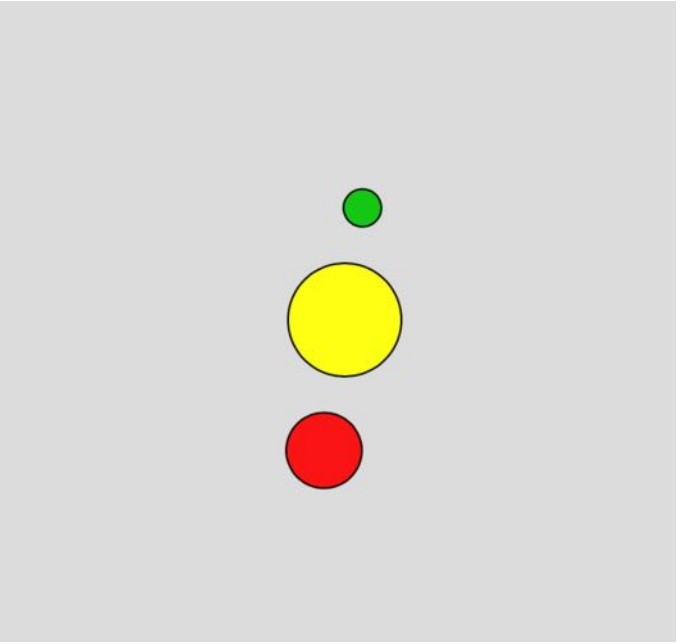
- To spin a shape around its center point at a place on screen away from the origin, first use translate() to move to the location where you'd like the shape, then call rotate(), and then draw the shape with its center at coordinate (0,0):



```
var angle = 0.0;

function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  rotate(angle);
  rect(-15, -15, 30, 30);
  angle += 0.1;
}
```

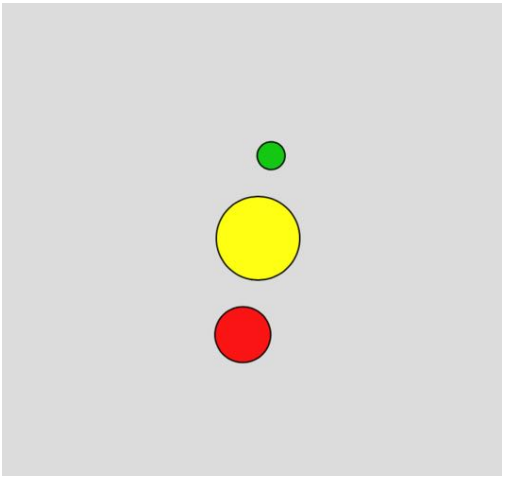
Practice: Solar System

- Use the translate and rotate options to draw a simple solar system
  - Rotate at least two planets around the Sun



Practice: Solar System

- Use the translate and rotate options to draw a simple solar system
  - Rotate at least two planets around the Sun

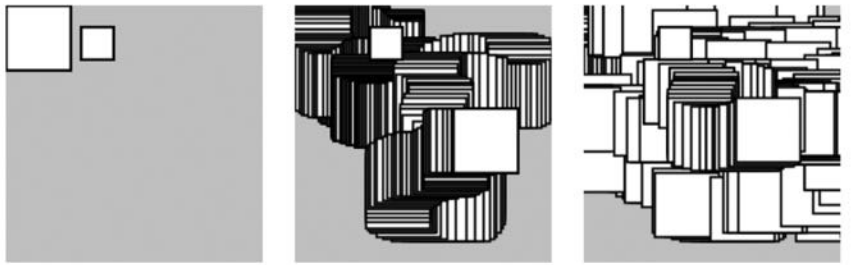


```
function setup() {
  createCanvas(400, 400);
  angleMode(DEGREES);
}

angle = 0

function draw() {
  background(220);

  translate(width/2, height/2);
  rotate(angle);
  fill (255, 255, 20);
  ellipse(0, 0, 60, 60);

  translate(60, 0);
  fill (20, 200, 20);
  ellipse(0,0, 20 ,20)


  translate(-130, 0);
  fill (250, 20, 20);
  ellipse(0,0, 40 ,40)

  angle = angle + 1
}
```

Isolating Transformations: push() and pop() functions

- To isolate the effects of transformations so they don't affect later functions, use the push() and pop() functions.
- When push() is run, it saves a copy of the current coordinate system and then restores that system after pop().
- This is useful when transformations are needed for one shape, but not wanted for another.

- In this example, the smaller rectangle always draws in the same position because the translate(mouseX, mouseY) is cancelled by the pop():



```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  push();
  translate(mouseX, mouseY);
  rect(0, 0, 30, 30);
  pop();
  translate(35, 10);
  rect(0, 0, 15, 15);
}
```

Media

Arranging the graphics elements on screen

Media: Background

- p5.js is capable of drawing more than simple lines and shapes.
- Let us learn how to create images and text in our programs to extend the visual possibilities to photography, detailed diagrams, and diverse typefaces.

Images

- There are three steps to follow before you can draw an image to the screen:

- 1. Add the image to the sketch's folder.
- 2. Create a variable to store the image.
- 3. Load the image into the variable with loadImage().

- Download images from
  - http://p5js.org/learn/books/media.zip

Load an Image

- In order to load an image, we will introduce a new function called preload().
- The preload() function runs once before the setup() function runs.
- You should generally load your images and other media in preload() in order to ensure they are fully loaded before your program starts.

Drawing an Image

- After all three steps are done, you can draw the image to the screen with the image() function.
- The first parameter to image() specifies the image to draw; the second and third set the x and y coordinates:

```
var img;

function preload() {
  img = loadImage("lunar.jpg");
}

function setup() {
  createCanvas(480, 120);
}

function draw() {
  image(img, 0, 0);
}
```

Load More Images

- For this example, you'll need to add the capsule.jpg file (found in the media folder you downloaded) to your sketch folder:

```
var img1;
var img2;

function preload() {
  img1 = loadImage("lunar.jpg");
  img2 = loadImage("capsule.jpg");
}

function setup() {
  createCanvas(480, 120);
}

function draw() {
  image(img1, -120, 0);
  image(img1, 130, 0, 240, 120);
  image(img2, 300, 0, 240, 120);
}
```

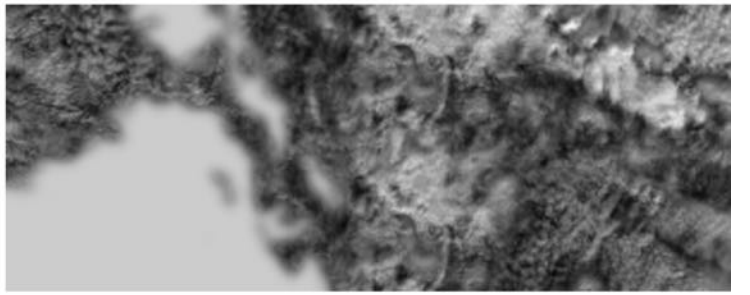
Mousing Around with Images

- When the mouseX and mouseY values are used as part of the fourth and fifth parameters of image(), the image size changes as the mouse moves:

```
var img;

function preload() {
  img = loadImage("lunar.jpg");
}

function setup() {
  createCanvas(480, 120);
}

function draw() {

 background(0);
 image(img, 0, 0, mouseX * 2, mouseY * 2);
}
```

Transparency with a PNG

- GIF, PNG, and SVG images support transparency, which means that pixels can be invisible or partially visible



```
var img;

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

function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  image(img, 0, 0);
  image(img, 0, mouseY * -1);
}
```

Text Stroke and Fill

- Just like shapes, the text is affected by both the stroke() and fill() functions.
- The following example outputs black text with a white outline:


```
function setup() {
  createCanvas(480, 120);
  textFont("Source Code Pro");
  fill(0);
  stroke(255);
}

function draw() {
  background(102);
  textSize(28);
  text("one small step for man...", 25, 60);
  textSize(16);
  text("one small step for man...", 27, 90);
}
```

Timer in the Game

- The game should end within a given time
- When the time is reached the game will display a message and stop
  - You might want to provide option so that the game can be restarted at this point

Timer in the Game: code

- The game should end within a given time
- When the time is reached the game will display a message and stop
  - You might want to provide option so that the game can be restarted at this point

```
var timeCount = 0;
var totalTime = 10;

function setup() {
  createCanvas(400, 400);
  textSize(15);
}

function draw() {
  background(20, 200, 240);

  timeCount = round(millis()/1000);

  text("Time: " + timeCount, 20, 20);

  if(timeCount >= 10){
    stroke(255);
    fill(240, 20, 20);
    textSize(20);
    text("Time ran out ...", 80, 180);
    noLoop();
  }
}
```

Timer in the Game with Restart Option

- If we want to provide restart option then we need to use some additional variables

- For example, there should be one variable that will allow or disallow the game play code to execute

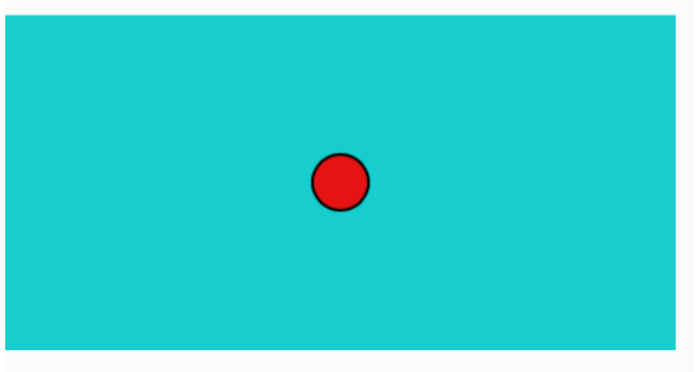
Timer in the Game with Restart Option

- When the user clicks on the restart button, we have to record that time and start the time counting from that position
- We can achieve this by using
  - timeCount = millis() - timePrev

```
var timeCount = 0, timePrev=0;
var totalTime = 10;
var gamePlay = 1;

function draw() {
  background(20, 200, 240);

  if (gamePlay == 1) {
    timeCount = round( (millis() - timePrev) / 1000);

    text("Time: " + timeCount, 20, 20);
  }

  if (timeCount >= 10) {
    text("Time ran out ...", 80, 180);
    gamePlay = 0;

    rect(80, 210, 100, 30);
    text("Restart game", 85, 225);
  }

  if(gamePlay ==0 && mouseIsPressed){
    if(mouseX>80 && mouseX<180 && mouseY>210 && mouseY<240){
      gamePlay = 1;
      timePrev = millis();
    }
  }
}
```

Bouncing Ball

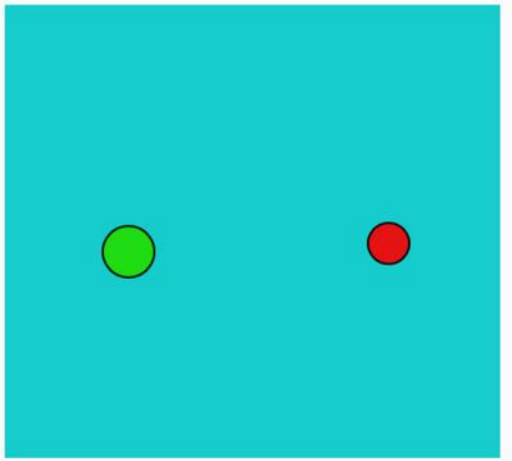
- Sometimes you want to animate a shape within a given boundary.
- With a few lines of code, you can set up the start position and specify a boundary within which the shape is going to animate



```
var step = 2;
var p = step, q = step;
var x = 0, y = 0;

function setup() {
  createCanvas(240, 120);
  background(20, 255, 255);
  x = width / 2;
  y = height / 2;
}

function draw() {
  background(20, 205, 205);

  if (x >= width) p = -step;
  if (x <= 0) p = step;
  if (y >= height) q = -step;
  if (y <= 0) q = step;

  x += p;
  y += q;

  fill(230, 20, 20);
  ellipse(x, y, 20, 20);
}
```

Practice: Two Bouncing Balls

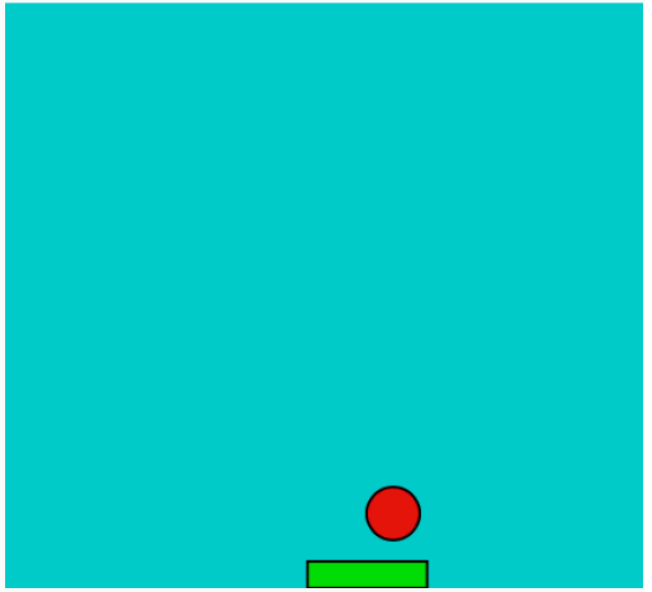
- Modify the previous example and add two bouncing balls that are going to animate within the given boundary



```
var step = 2;
var p = step, q = step;
var x = 0, y = 0;

function setup() {
  createCanvas(240, 120);
  background(20, 255, 255);
  x = width / 2;
  y = height / 2;
}

function draw() {
  background(20, 205, 205);

  if (x >= width) p = -step;
  if (x <= 0) p = step;
  if (y >= height) q = -step;
  if (y <= 0) q = step;

  x += p;
  y += q;

  fill(230, 20, 20);
  ellipse(x, y, 20, 20);
}
```

Assignment: Bouncing Ball and a Paddle

- Modify the example, so that the ball will bounce off only when there is a paddle at the bottom
- When the cy value of the ball reaches,
  - if(cy >= height - 10)
  - then, check whether there is an intersection with the paddle
  - If there is an intersection, then change the q = -step
  - Otherwise, display "You have Lost the Game"



Multimedia: playing sound

- Adding a sound and playing it at certain event can enhance the game experience a lot
- In order to add sound, we have to do the following:
    - 1. Add the sound to the sketch's folder.
    - 2. Create a variable to store the sound.
    - 3. Load the sound into the variable with loadSound().
- Download sound files from
    - https://freesound.org/browse/

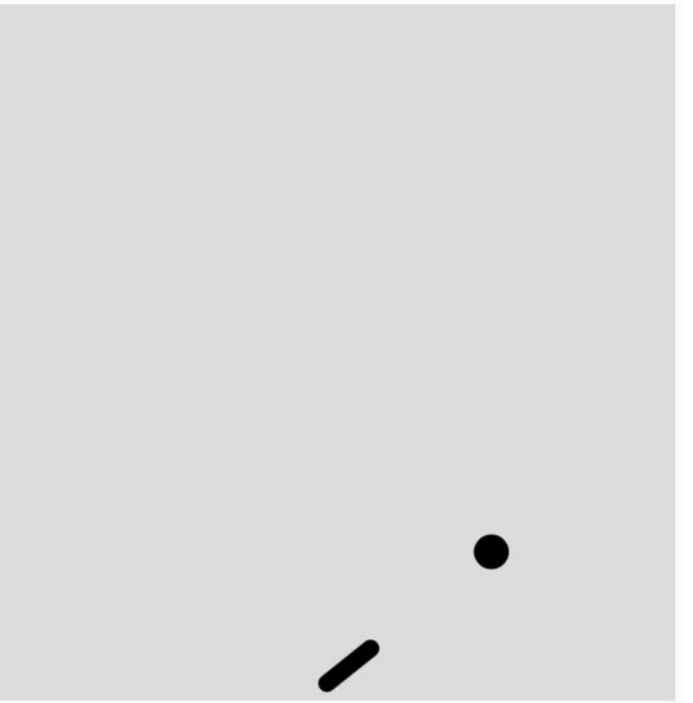
Multimedia: playing sound

- In order to start the sound, use song.play() function
- In order to stop the sound, use song.stop() function
- You can also use song.loop() function to make the sound repeat in a loop
- You can add as many sound files as needed
  - however, you must use them creatively to add to the experience of your application

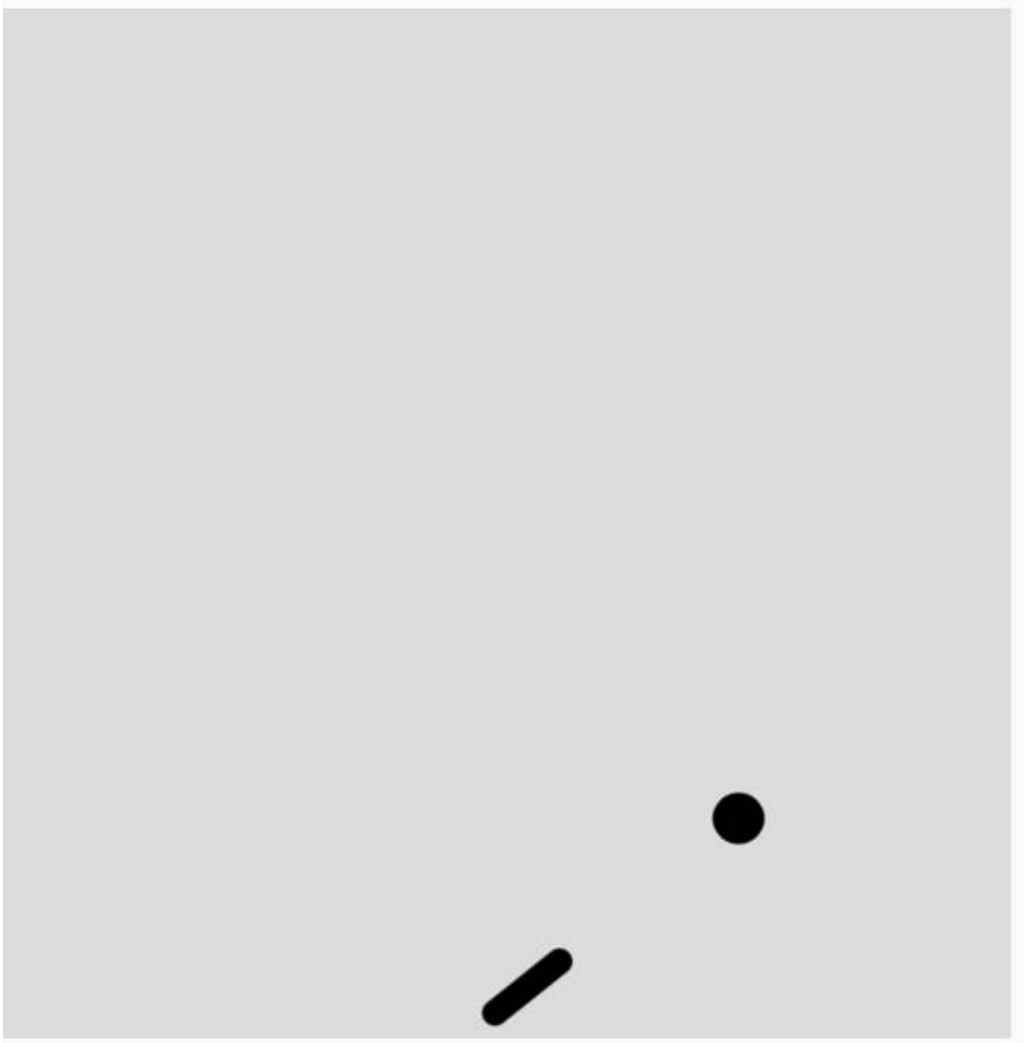
```
var song;

function preload() {
  song = loadSound("ambient.mp3");
}

function setup() {
  createCanvas(400, 400);
  textSize(15);
  song.play();
  song.loop();
}
```

Canon

- We want to implement the behavior of a canon and use it in our project
- The idea is to shoot the ball by using the canon when mouse is pressed at the direction pointed at by the canon



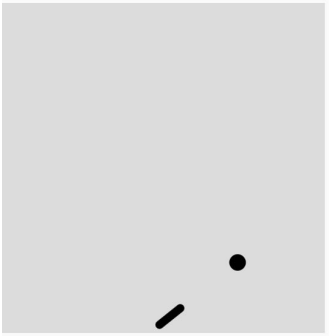
Canon: line calculation



Canon: initial setup

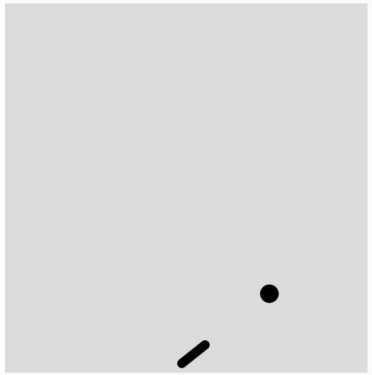
- We want to implement the behavior of a canon and use it in our project
- Let us create some variables and initialize them in the setup function
- ballx, bally variables hold the position of the bullet/ball
- rx is the position of the canon
- m variable holds the slope of the line
- b is the offset part of the line

```
var rx = 0; // canon position
var ballx = 0;
var bally = 0;

var m = 1; // slope
var b = 0; // y = mx + b

function setup() {
  createCanvas(400, 400);
  ballx = width;
  bally = -20; // initial position of the ball
  step = 5;
}
```



Canon: calculating the ball direction and position

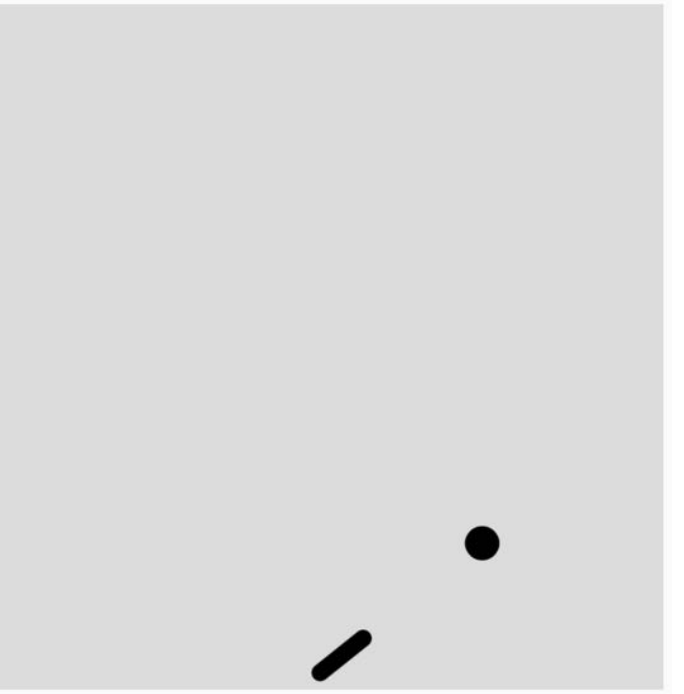
- When mouseIsPressed, we calculate the slope of the canon
- we calculate the initial position of the ball (at the end of the canon)
- We calculate the offset (b) value of the line
- By using slope m and the offset b, we calculate the x position of the ball as the y value decrease by 5 pixels.



```
if (mouseIsPressed) {
  // m = (y1-y2)/(x1-x2)
  var x1 = width / 2;
  var x2 = rx;
  var y1 = height - 10;
  var y2 = height - 30;
  // can not divide by zero (0)
  if (x1 - x2 != 0) {
    m = (y1 - y2) / (x1 - x2);
  } else m = 1;

  ballx = rx; // initial ball position
  bally = height - 30;
  b = bally - m * ballx; // value of b
}

bally = bally - step; // ball moving upward
ballx = (bally - b) / m; // calculating ball x

ellipse(ballx, bally, 10, 10); // displaying ball
}
```

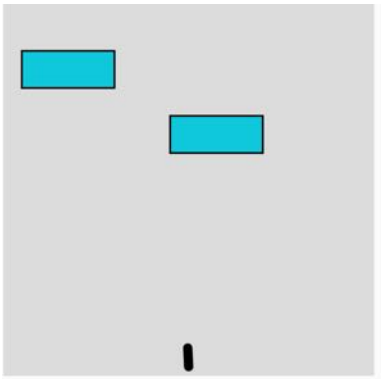
Canon: calculating the ball direction and position

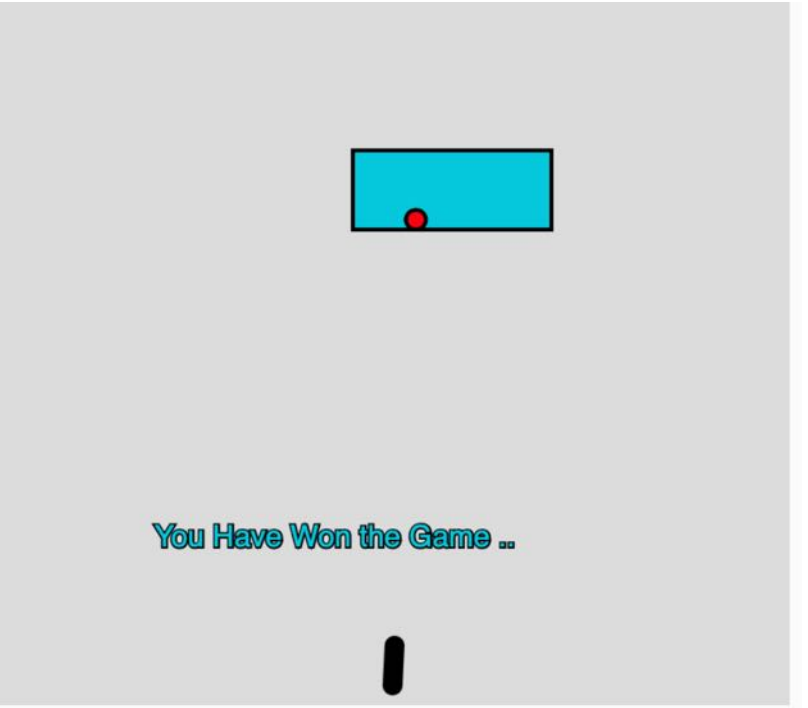
- The complete project can be accessed at the following link:
  - HyperLink



Practice: destroy one or two rectangles by using the canon

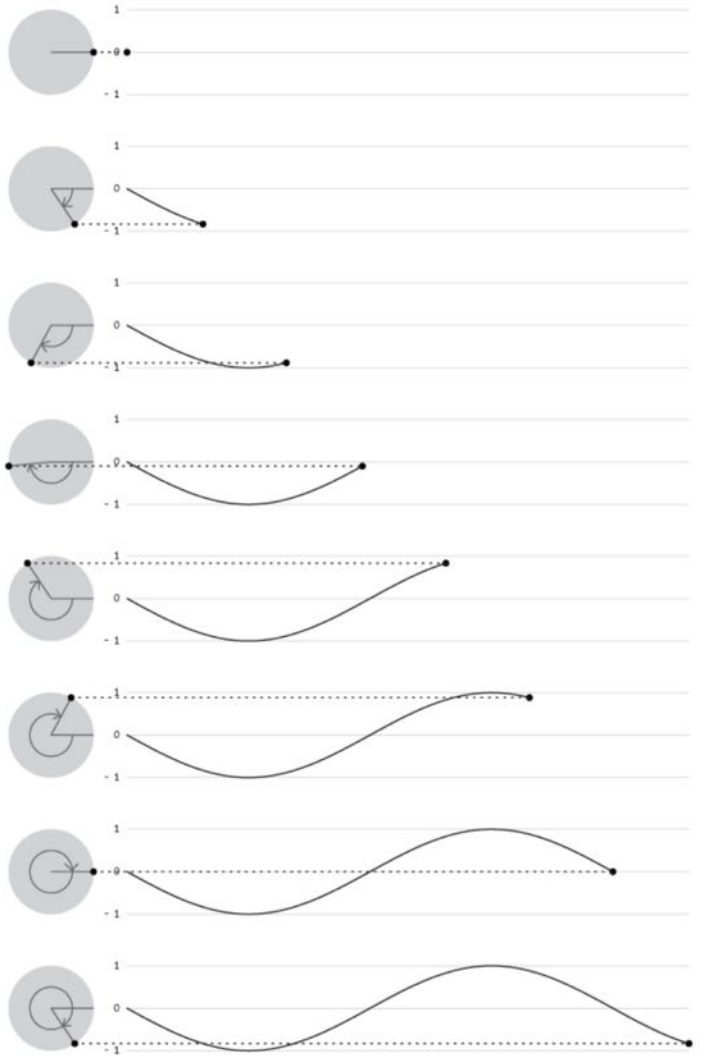
- Additional challenge:
  - canon ball will bounce off if it hits the side walls





Circular: visualization

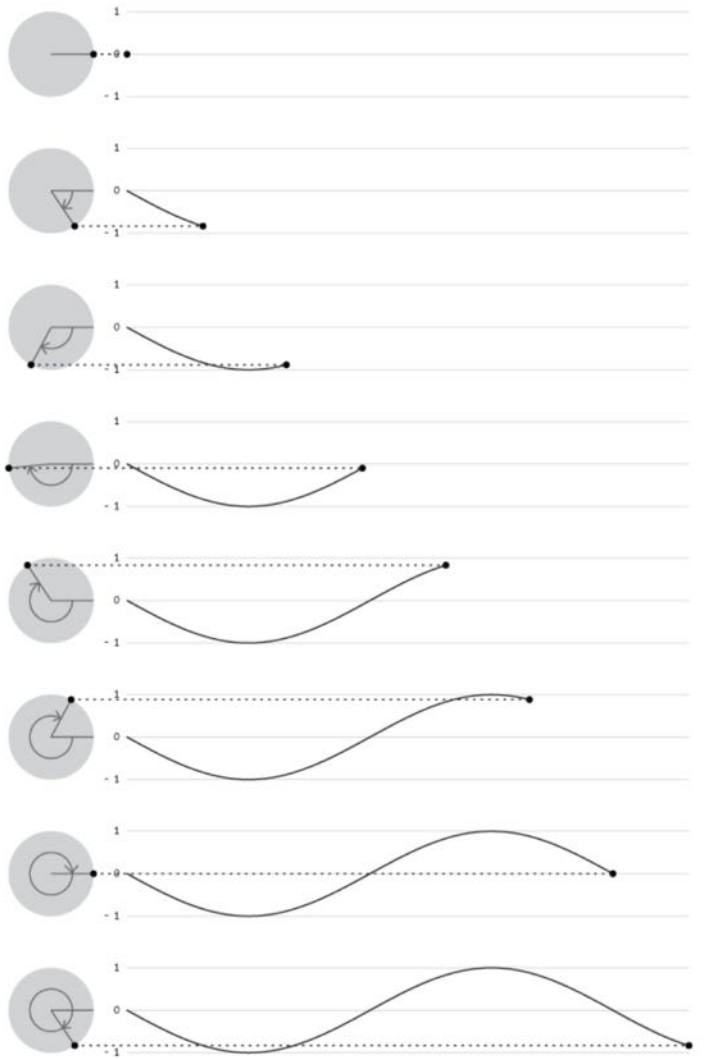
- If you're a trigonometry ace, you already know how amazing the sine and cosine functions are.
- If you're not, we hope the next examples will trigger your interest.
- The figure shows a visualization of sine wave values and how they relate to angles.
  - At the top and bottom of the wave, notice how the rate of change (the change on the vertical axis) slows down, stops, then switches direction.
  - It's this quality of the curve that generates interesting motion.



Circular: mapping the values

- The sin() and cos() functions in p5.js return values between −1 and 1 for the sine or cosine of the specified angle.

- To be useful for drawing, the float values returned by sin() and cos() are usually multiplied by a larger value.



Circular: Sine Wave Values

- This example shows how values for sin() cycle from −1 to 1 as the angle increases.
- With the map() function, the sinval variable is converted from this range to values from 0 to 255.
- This new value is used to set the background color of the canvas:

```
var angle = 0.0;

function draw() {
  var sinval = sin(angle);
  print(sinval);
  var gray = map(sinval, -1, 1, 0, 255);
  background(gray);
  angle += 0.1;
}
```

- Note: if you are using angleMode(DEGREES) then increase the value of the angle as, angle = angle + 1

Circular: Sine Wave Movement

- This example shows how the generated sine values can be converted into movement:

```
var angle = 0.0;
var offset = 60;
var scalar = 40;

var ballSpeed = 0.05;

function setup() {
  createCanvas(240, 120);
}

function draw() {
  background(0);
  var y1 = offset + sin(angle) * scalar;
  var y2 = offset + sin(angle + 0.4) * scalar;
  var y3 = offset + sin(angle + 0.8) * scalar;
  ellipse(80, y1, 40, 40);
  ellipse(120, y2, 40, 40);
  ellipse(160, y3, 40, 40);
  angle += ballSpeed;
}
```

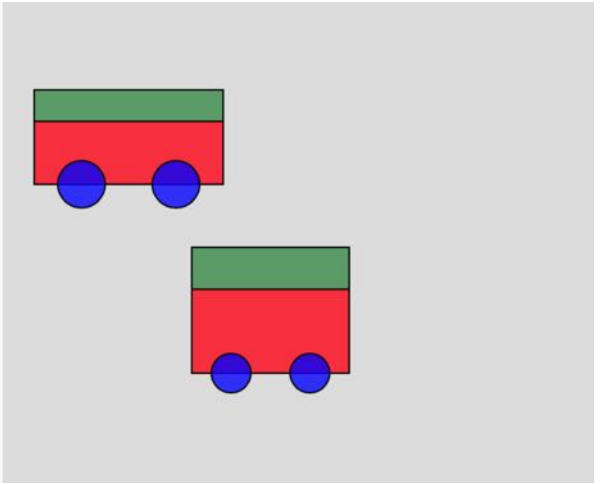
Practice: Follow the Sine Wave Movement

- Create three ellipses and make it follow the sine movement
- For each ellipse, increase the value of x1, x2, x3 and use the generated y1, y2, y3 values respectively to draw the balls

```
var angle = 0.0;
var offset = 60;
var scalar = 40;

var ballSpeed = 0.05;

function setup() {
  createCanvas(240, 120);
}

function draw() {
  background(0);
  var y1 = offset + sin(angle) * scalar;
  var y2 = offset + sin(angle + 0.4) * scalar;
  var y3 = offset + sin(angle + 0.8) * scalar;
  ellipse(80, y1, 40, 40);
  ellipse(120, y2, 40, 40);
  ellipse(160, y3, 40, 40);
  angle += ballSpeed;
}
```

Function with Parameters

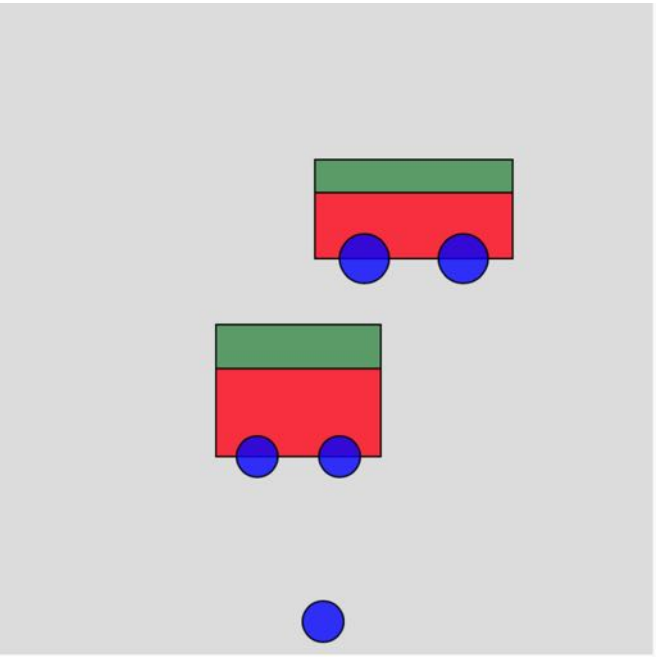
- We can rewrite the function that can be called by specifying different values for the function parameters



```
function carParam(cx, cy, w, h){

  fill(255, 0, 20, 200); // body
  rect(cx, cy, w, h);

  fill(25, 200, 120, 180); // windshield
  rect (cx, cy, w, h/3);

  var r = w/4;

  fill(0, 0, 255, 200); // left wheel
  ellipse(cx + r, cy + h, r, r);

  fill(0, 0, 255, 200); // right wheel
  ellipse(cx + w - r, cy + h, r, r);
}


function draw() {
  background(220);
  // first car
  carParam(20, 100, 120, 60);
  // another car
  carParam(120, 200, 100, 80);
}
```

Assignment: Crossy Roads

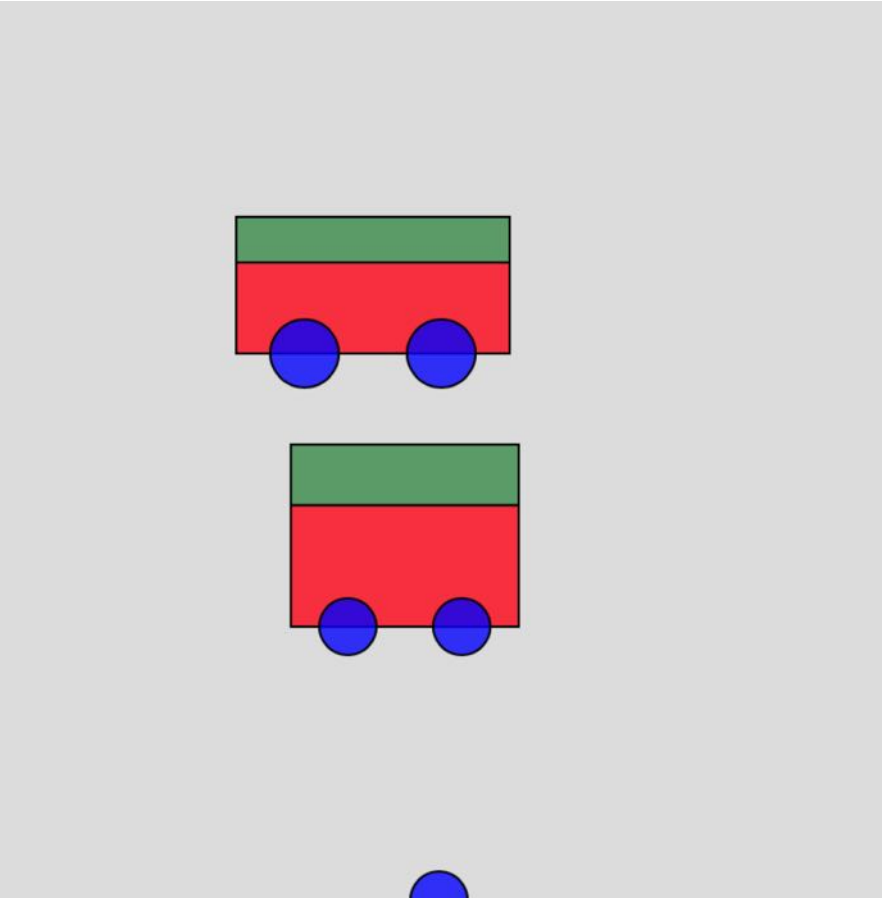
- Use the carParam function and draw four more cars on the canvas
  - make sure they are of different sizes and placed at different locations
- Now, draw an ellipse as the user and try to cross the road
- Use the keyboard arrow keys to control the user ellipse
- When detecting intersection, treat the car as if it is a rectangle (do not consider the wheels)



- You can use the template code found at this link

Project Link

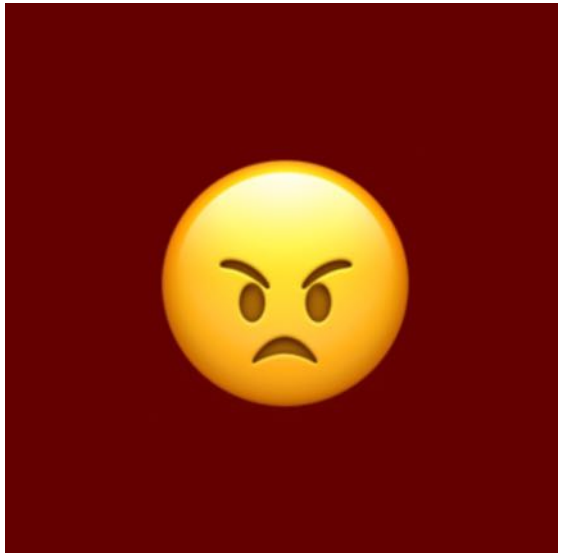
- The basic CodyRoads game template can be found at this link



Mouse & Keyboard

Interesting: Angry Emoji

- The computer gets mad when you click on the canvas



Interesting: Angry Emoji

- The computer gets mad when you click on the canvas

```
let isMad = false;

function setup() {
  createCanvas(300, 300);
  textAlign(CENTER, CENTER);
  textSize(144);
}

function draw() {
  if (isMad) {
    background(random(64, 255), 0, 0);
    text("😠", width / 2 + random(-10, 10), height / 2 +
random(-10, 10));
  } else {
    background(64);
    text("🙂", width / 2, height / 2);
  }
}

function mousePressed() {
  isMad = !isMad;
}
```

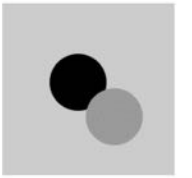
Interesting: Angry Emoji

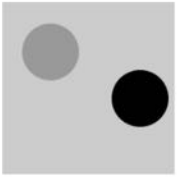
- Add other emojis. Make your computer laugh instead of getting mad!
- Switch between related emojis, like 🐵🙈🙉🙊🐒!

```
let isMad = false;

function setup() {
  createCanvas(300, 300);
  textAlign(CENTER, CENTER);
  textSize(144);
}

function draw() {
  if (isMad) {
    background(random(64, 255), 0, 0);
    text("😠", width / 2 + random(-10, 10), height / 2 +
random(-10, 10));
  } else {
    background(64);
    text("🙂", width / 2, height / 2);
  }
}

function mousePressed() {
  isMad = !isMad;
}
```

Mouse Interactivity

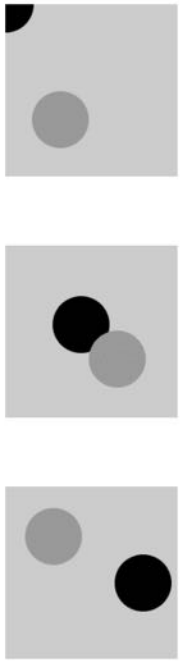
```
var dragX=0, dragY=0, moveX=0, moveY=0;

function setup() {
  createCanvas(100, 100);
  noStroke();
}

function draw() {
  background(204);
  fill(0);
  ellipse(dragX, dragY, 33, 33); // Black circle
  fill(153);
  ellipse(moveX, moveY, 33, 33); // Gray circle
}

function mouseMoved() { // Move gray circle
  moveX = mouseX;
  moveY = mouseY;
}

function mouseDragged() { // Move black circle
  dragX = mouseX;
  dragY = mouseY;
}
```





Practice: Mouse Interactivity

- Display a normal emoji by using the emoji function
- Drag the mouse cursor to move the emoji at a new location

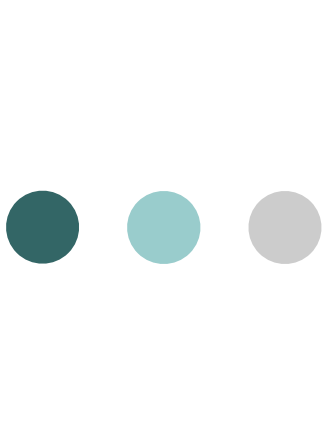


KeyBoard Interactivity

```
var drawT = false;

function setup() {
  createCanvas(100, 100)
  noStroke();
}

function draw() {
  background(204);
  if (drawT == true) {
    rect(20, 20, 60, 20);
    rect(39, 40, 22, 45);
  }
}

function keyPressed() {
  if ((key == 'T') || (key == 't')) {
    drawT = true;
  }
}

function keyReleased() {
  drawT = false;
}
```

Practice: KeyBoard Interactivity

- Display a normal emoji by using the emoji function
- When any key is pressed, change the emoji to a happy emoji
- If no key is pressed display the normal emoji again



Array

Array

- An array is a list of variables that share a common name.
- Arrays are useful because they make it possible to work with more variables without creating a new name for each one.
- This makes the code shorter, easier to read, and more convenient to update.

Array: Many Variables

- In this example, we demonstrate the usefulness of an array

```
var x1 = -20;
var x2 = 20;

function setup() {
  createCanvas(240, 120);
  noStroke();
}

function draw() {
  background(0);
  x1 += 0.5;
  x2 += 0.5;
  arc(x1, 30, 40, 40, 0.52, 5.76);
  arc(x2, 90, 40, 40, 0.52, 5.76);
}
```

Array: Too Many Variables

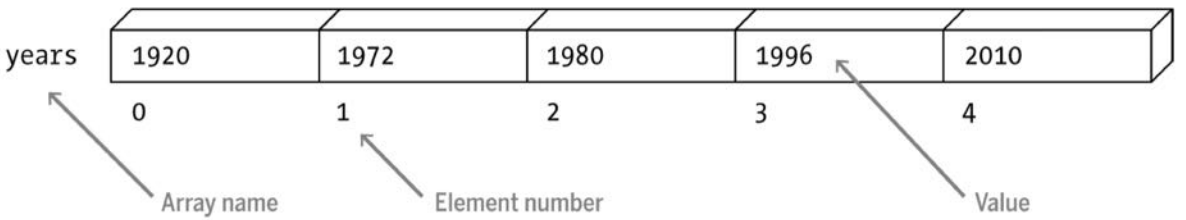
- The code for the previous example is still manageable, but what if we want to have five circles?
- We need to add three more variables to the two we already have:

```
var x1 = -10;
var x2 = 10;
var x3 = 35;
var x4 = 18;
var x5 = 30;

function setup() {
  createCanvas(240, 120);
  noStroke();
}

function draw() {
  background(0);
  x1 += 0.5;
  x2 += 0.5;
  x3 += 0.5;
  x4 += 0.5;
  x5 += 0.5;
  arc(x1, 20, 20, 20, 0.52, 5.76);
  arc(x2, 40, 20, 20, 0.52, 5.76);
  arc(x3, 60, 20, 20, 0.52, 5.76);
  arc(x4, 80, 20, 20, 0.52, 5.76);
  arc(x5, 100, 20, 20, 0.52, 5.76);
}
```

Make an Array

- An array is a list of one or more variables that share the same name
- Using arrays is similar to working with single variables; it follows the same patterns. As you know, you can make a single variable called x with this code: var x
- To make an array, just set the variable's value to a set of empty brackets: var x = [];
  - Note that the length of the array does not need to be declared in advance; The length is determined by the number of elements you put into it.

```
var years = [ 1920, 1972, 1980, 1996, 2010 ];
```



Make an Array

- Each item in an array is called an element, and each has an index value to mark its position within the array.
- Just like coordinates on the canvas, index values for an array start counting from 0.
  - For instance, the first element in the array has the index value 0, the second element in the array has the index value 1, and so on.
  - If there are 20 values in the array, the index value of the last element is 19.

Declare and Assign an Array

- First, we'll declare the array outside of setup() and then create and assign the values within.
- The syntax x[0] refers to the first element in the array and x[1] is the second:

```
var x = [];          // Declare the array

function setup() {
  createCanvas(200, 200);
  x[0] = 12;         // Assign the first value
  x[1] = 2;          // Assign the second value
}
```

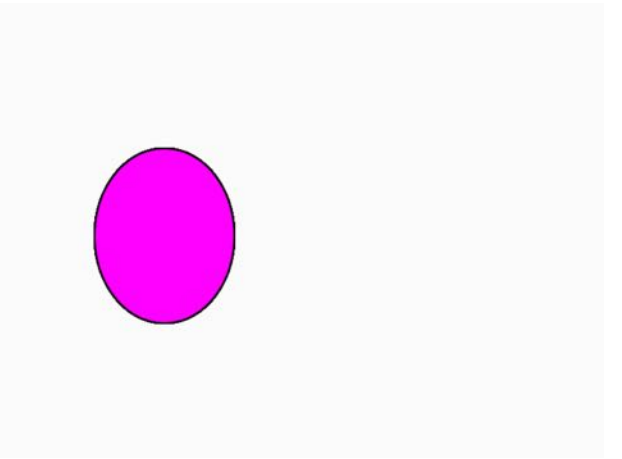
Declare and Assign an Array

- You can also assign values to the array when it's created, if it's all part of a single statement:

```
var x = [12, 2]; // Declare and assign

function setup() {
  createCanvas(200, 200);
}
```

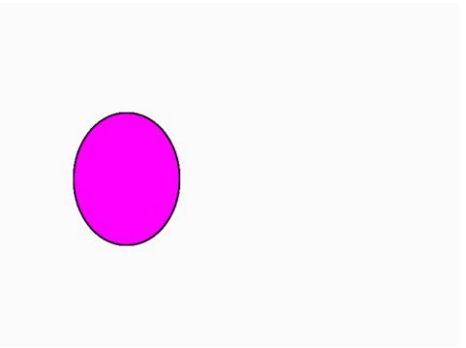
Using Colors from Array

- Use array elements to set color for a set of drawing objects
  - use a randomly generated index to access the color values stored in the array



Using Colors from Array

- Use array elements to set color for a set of drawing objects
  - use a randomly generated index to access the color values stored in the array



```
var colorNames = [
  "red",
  "green",
  "blue",
  "cyan",
  "magenta",
  "white",
  "purple",
  "yellow",
  "black",
];

let selectColorIndex = 0;
function setup() {
  createCanvas(400, 400);
  selectColorIndex = round(random(0, colorNames.length-1));
}

function draw() {
  if (mouseIsPressed) {
    selectColorIndex = round(random(0, colorNames.length-
1));
  }

  fill(colorNames[selectColorIndex]);
  ellipse(100, 200, 80, 100);

}
```

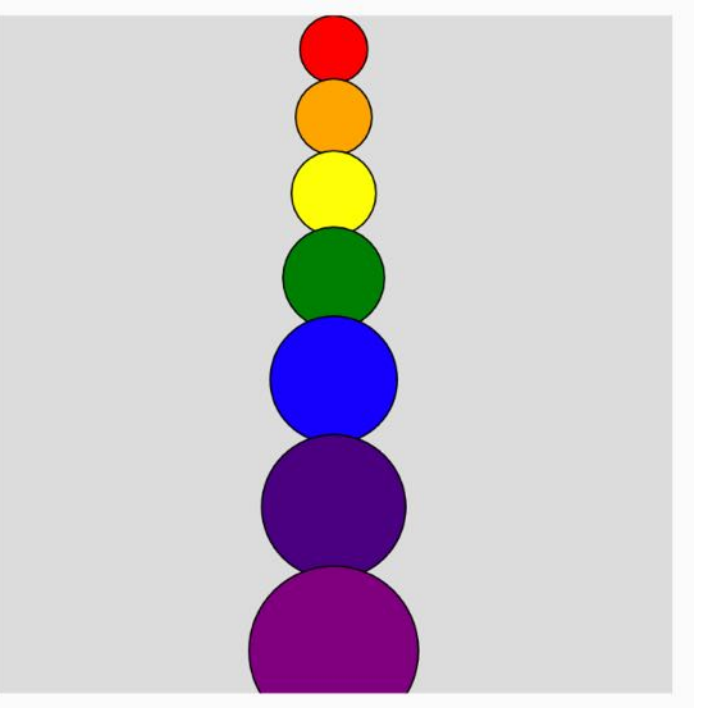
Using Colors from Array in a Loop

- In this example, we are using a loop to access each color starting from the 0 index
- For each color, we are drawing an ellipse on the canvas

```
var colorNames = [
  "red",
  "green",
  "blue",
  "cyan",
  "magenta",
  "white",
  "purple",
  "yellow",
  "black",
];

let selectColorIndex = 0;
function setup() {
  createCanvas(400, 400);
  selectColorIndex = round(random(0, colorNames.length - 1));
}

function draw() {
  var i = 0;
  var y = 20;
  for (i = 0; i < colorNames.length; i++) {
    fill(colorNames[i]);
    ellipse(100, y, 25, 25);
    y = y + 28;
  }
}
```

Practice: Draw Snowman using Colors from an Array

- Use the idea presented in the previous example
- Make a rainbow snowman
  - Use one loop
  - Use the color names from the array



Array: Many Variables

- In this example, we demonstrate the usefulness of an array

```
var x = [-20, 20];

function setup() {
  createCanvas(240, 120);
  noStroke();
}

function draw() {
  background(0);
  x[0] += 0.5;  // Increase the first element
  x[1] += 0.5;  // Increase the second element
  arc(x[0], 30, 40, 40, 0.52, 5.76);
  arc(x[1], 90, 40, 40, 0.52, 5.76);
}
```

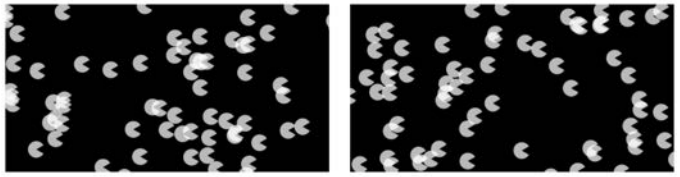
Practice: animate four shapes

- By extending the presented example, add four shapes and animate them on the canvas

```
var x = [-20, 20];

function setup() {
  createCanvas(240, 120);
  noStroke();
}

function draw() {
  background(0);
  x[0] += 0.5;  // Increase the first element
  x[1] += 0.5;  // Increase the second element
  arc(x[0], 30, 40, 40, 0.52, 5.76);
  arc(x[1], 90, 40, 40, 0.52, 5.76);
}
```

Arrays, Not Variables

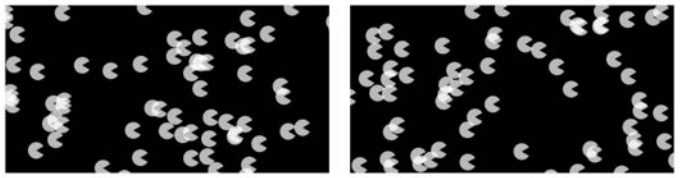
- Imagine what would happen if you wanted to have 3,000 circles.
- This would mean creating 3,000 individual variables, then updating each one separately.
  - Could you keep track of that many variables?
  - Would you want to?
- Instead, we use an array:



```
var x = [];

function setup() {
  createCanvas(240, 120);
  noStroke();
  fill(255, 200);
  for (var i = 0; i < 3000; i++) {
    x[i] = random(-1000, 200);
  }
}

function draw() {
  background(0);
  for (var i = 0; i < x.length; i++) {
    x[i] += 0.5;
    var y = i * 0.4;
    arc(x[i], y, 12, 12, 0.52, 5.76);
  }
}
```

Practice: Arrays, Not Variables

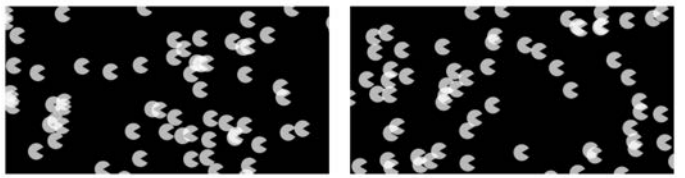
- They are all disappearing after a moment
  - how to put them back on the canvas again?



```
var x = [];

function setup() {
  createCanvas(240, 120);
  noStroke();
  fill(255, 200);
  for (var i = 0; i < 3000; i++) {
    x[i] = random(-1000, 200);
  }
}

function draw() {
  background(0);
  for (var i = 0; i < x.length; i++) {
    x[i] += 0.5;
    var y = i * 0.4;
    arc(x[i], y, 12, 12, 0.52, 5.76);
  }
}
```

Assignment: Arrays, Not Variables

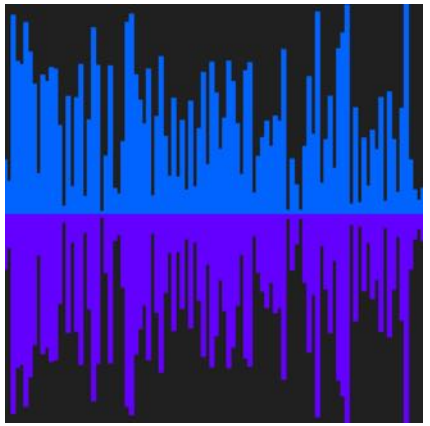
- Animate the arc by using sin curve
- Hint: x values are increasing (0 to width)
  - we have to calculate the y value for a given x[index]
  - y = sin (x[index]) * scalar



```
var x = [];

function setup() {
  createCanvas(240, 120);
  noStroke();
  fill(255, 200);
  for (var i = 0; i < 3000; i++) {
    x[i] = random(-1000, 200);
  }
}

function draw() {
  background(0);
  for (var i = 0; i < x.length; i++) {
    x[i] += 0.5;
    var y = i * 0.4;
    arc(x[i], y, 12, 12, 0.52, 5.76);
  }
}
```

Array: Oscillating Lines

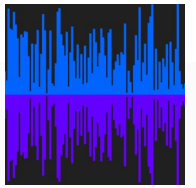
- This sketch uses an array to hold line height values, and then changes those values to show oscillating lines.



```
const lines = 100;
let lineHeights = [];
let lineSpeeds = [];

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

  for(let x = 0; x < lines; x++){
    const lineHeight = random(height);
    lineHeights[x] = lineHeight;
    lineSpeeds[x] = random(-10, 10);
  }
}
```

Array: Oscillating Lines

- Change the number of lines
- Give each line a random color (by using array).
- Make the animation look more like a mountain, or a city.
- You can use the equation:
  - var y = sin (lineHeights[i]) * scalar



```
const lines = 100;
let lineHeights = [];
let lineSpeeds = [];

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

  for(let x = 0; x < lines; x++){
    const lineHeight = random(height);
    lineHeights[x] = lineHeight;
    lineSpeeds[x] = random(-10, 10);
  }
}
```

```
function draw() {
  background(32);

  for(let i = 0; i < lines; i++){
    lineHeights[i] += lineSpeeds[i];

    if(lineHeights[i] < 0 || lineHeights[i] > height) {
      lineSpeeds[i] *= -1;
    }

    const x = width * (i / lines);
    const lineWidth = width / lines;

    fill(0, 100, 255);
    stroke(0, 100, 255);
    rect(x, height / 2 - lineHeights[i] / 2,
         lineWidth, lineHeights[i] / 2);

    fill(100, 0, 255);
    stroke(100, 0, 255);
    rect(x, height / 2,
         lineWidth, lineHeights[i] / 2);
  }
}
```

How to host your p5.js game on github pages

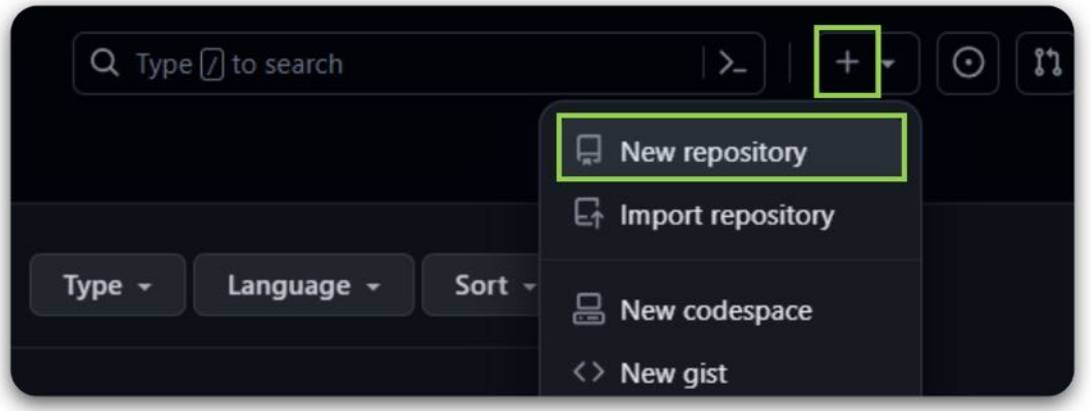
The following slides were created for this event by
– Shibam Pokhrel (CS Student, Truman)

Step 1

- Create a github account
  - https://github.com/signup

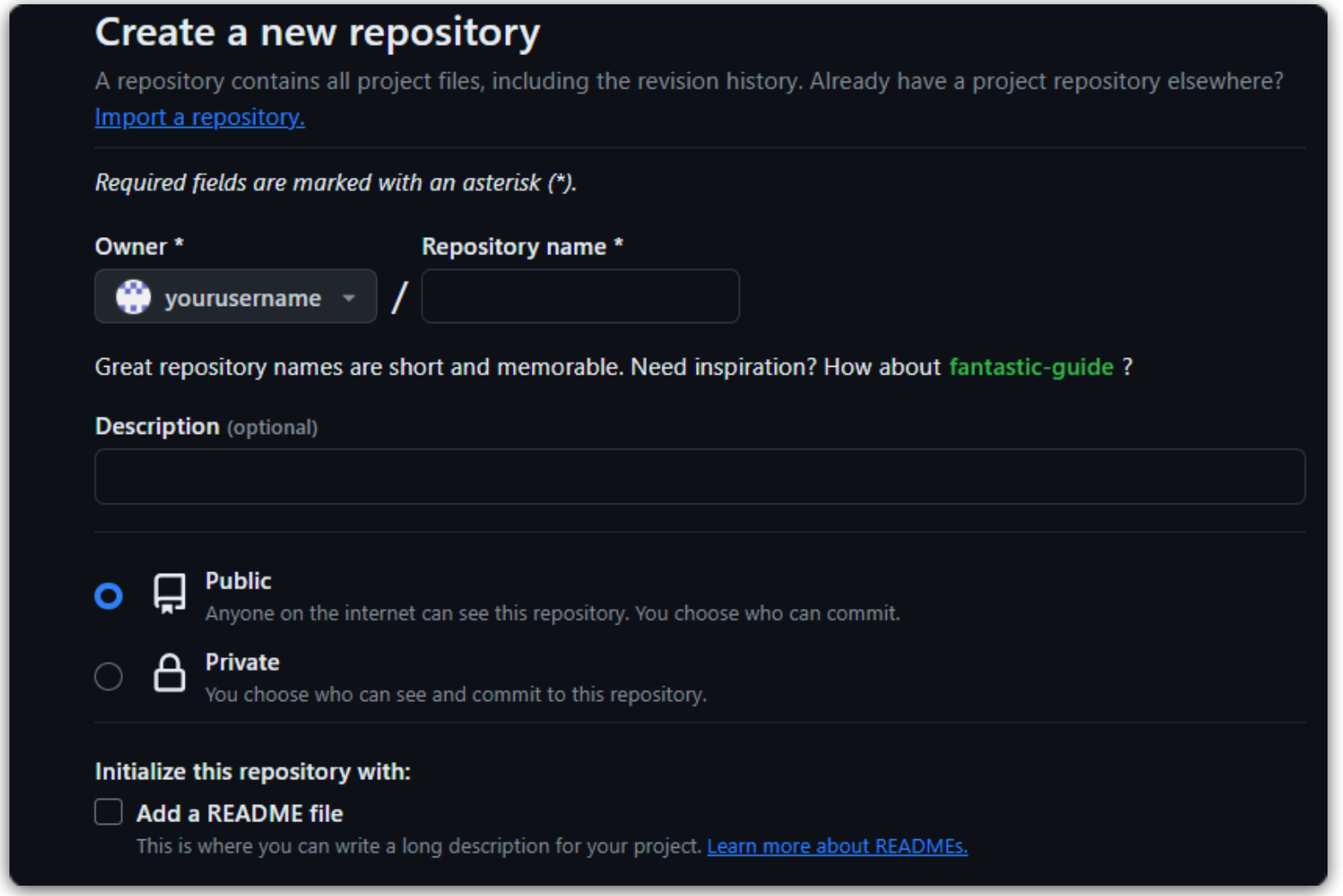
Step 2

- After logging into your github account, in the upper right corner of your screen, click on + icon and then select new repository
  - or you can go to github.com/new



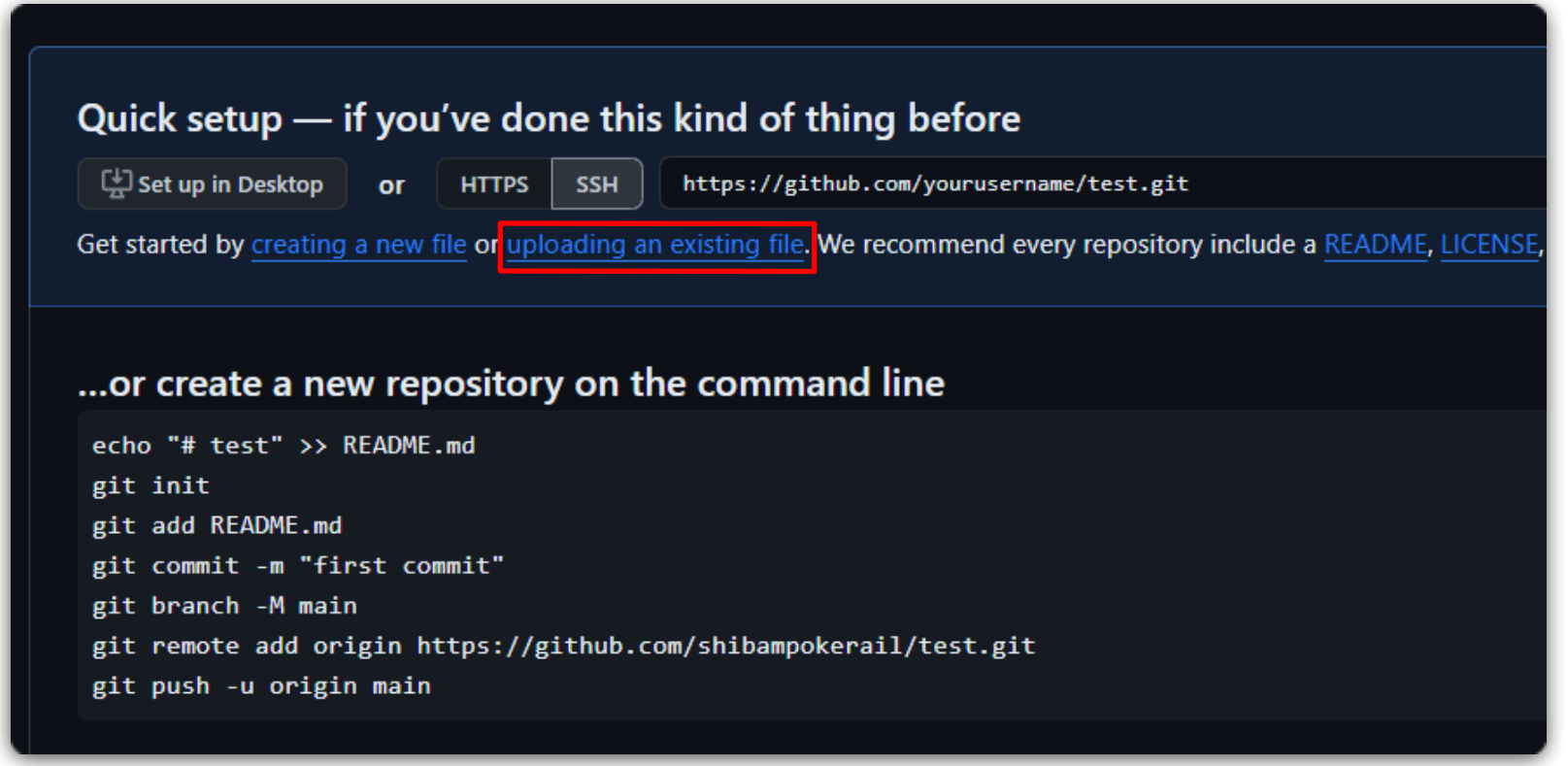
Step 3

- Create a new repository. (must be public)



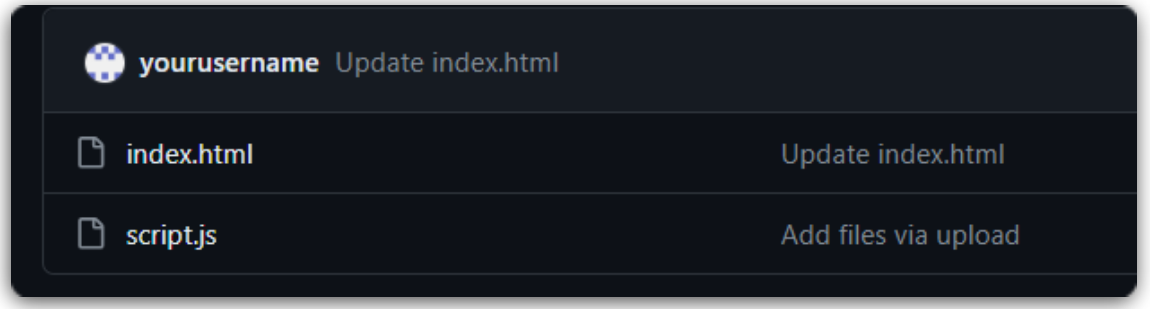
Step 4

- Migrate your local p5.js project to the repository. You can click on "uploading an existing file" to upload your p5.js script and html file
- OR use the console commands below to push your project to github but to do this you must have github installed locally.



Step 5

- Your root directory of your repository should look something like this.



- Your html file must be named "index.html". And there can be images if you are using any external images for your game.
- You should have at least,
  - an index.html
  - and a script.js file with your p5.js game script in your repository root.

Step 6

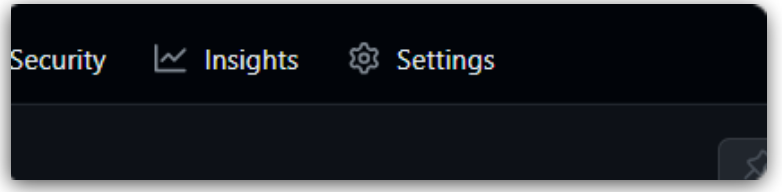
- The index.html should be something like this.
- You can add other stuffs to the index.html, as long as you have linked your p5.js script and imported the p5.js library
  - must import p5.js library inside the head tag by using the script tags
  - must link your index.html with your filename with your game code with the script tag at the end.

Code Blame 15 lines (14 loc) · 477 Bytes Code 55% faster with GitHub Copilot

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta http-equiv="X-UA-Compatible" content="IE=edge">
6       <meta name="viewport" content="width=device-width, initial-scale=1.0">
7       <title>Document</title>
8       <script src="https://cdn.jsdelivr.net/npm/p5@1.6.0/lib/p5.js"></script>
9       <script src="https://cdn.jsdelivr.net/npm/p5@1.6.0/lib/addons/p5.sound.js"></script>
10  </head>
11  <body>
12
13  </body>
14    <script src="script.js"></script>
15  </html>
```

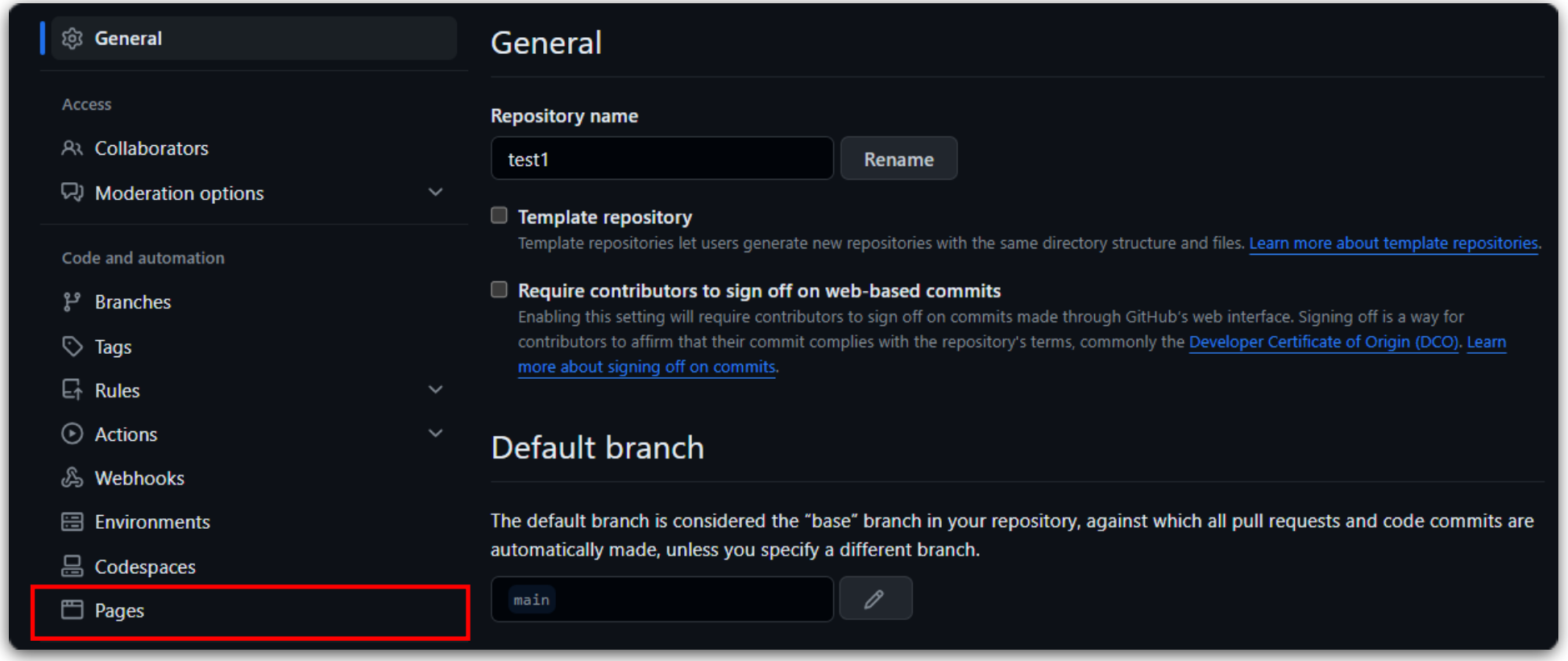
Step 7

- Then, click on the settings icon at the top middle of your screen (of your github repository).



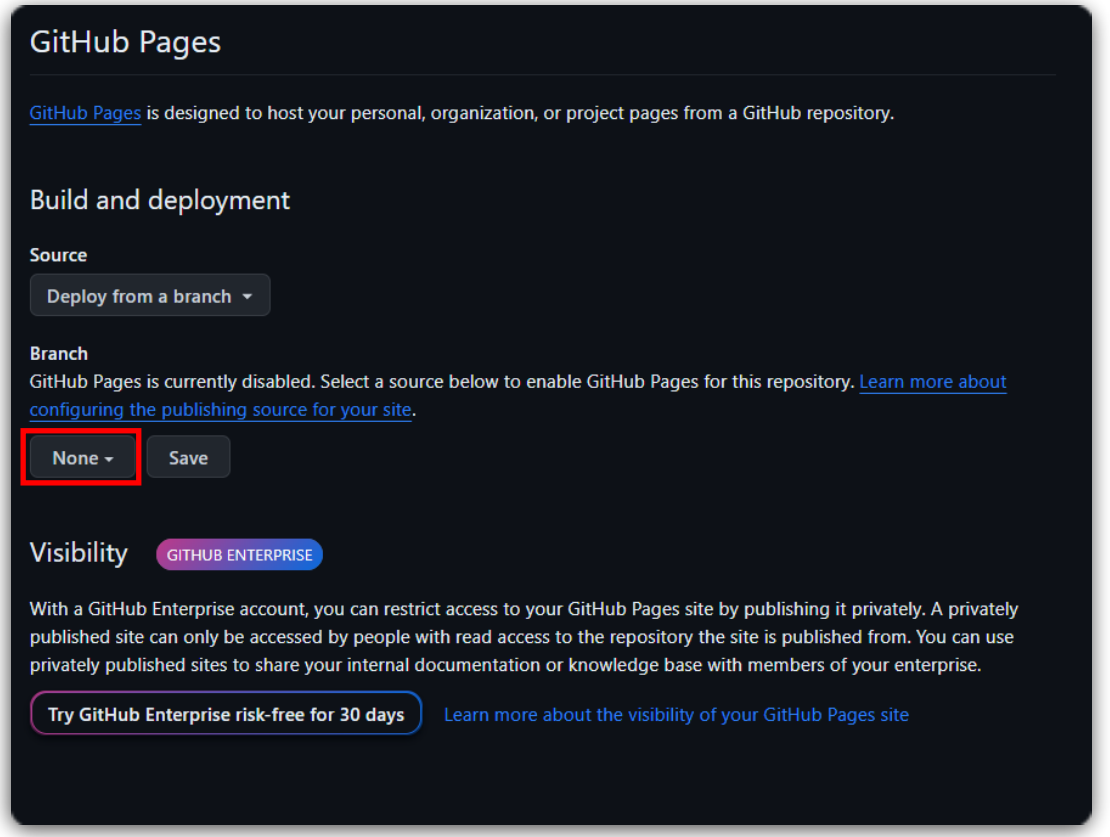
Step 8

- Then click on "pages" on the left sidebar, under "Code and automation".



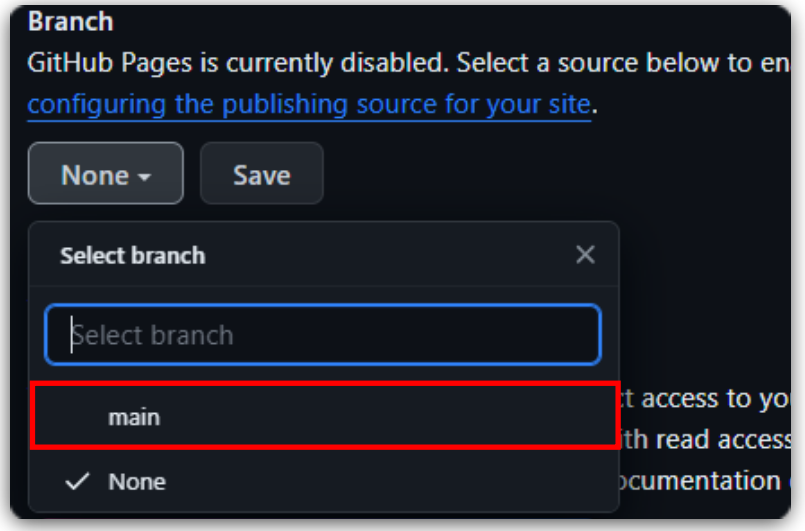
Step 9

- Then, under Branch, click on the dropdown option that says "None"

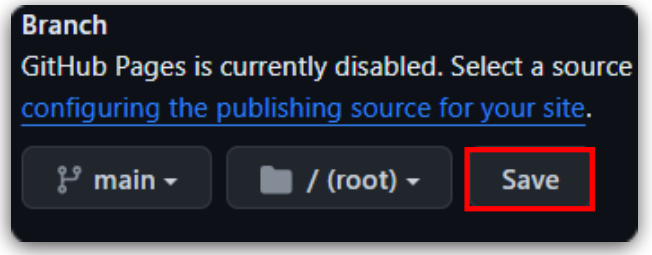


Step 10

- Select "main" from the dropdown.

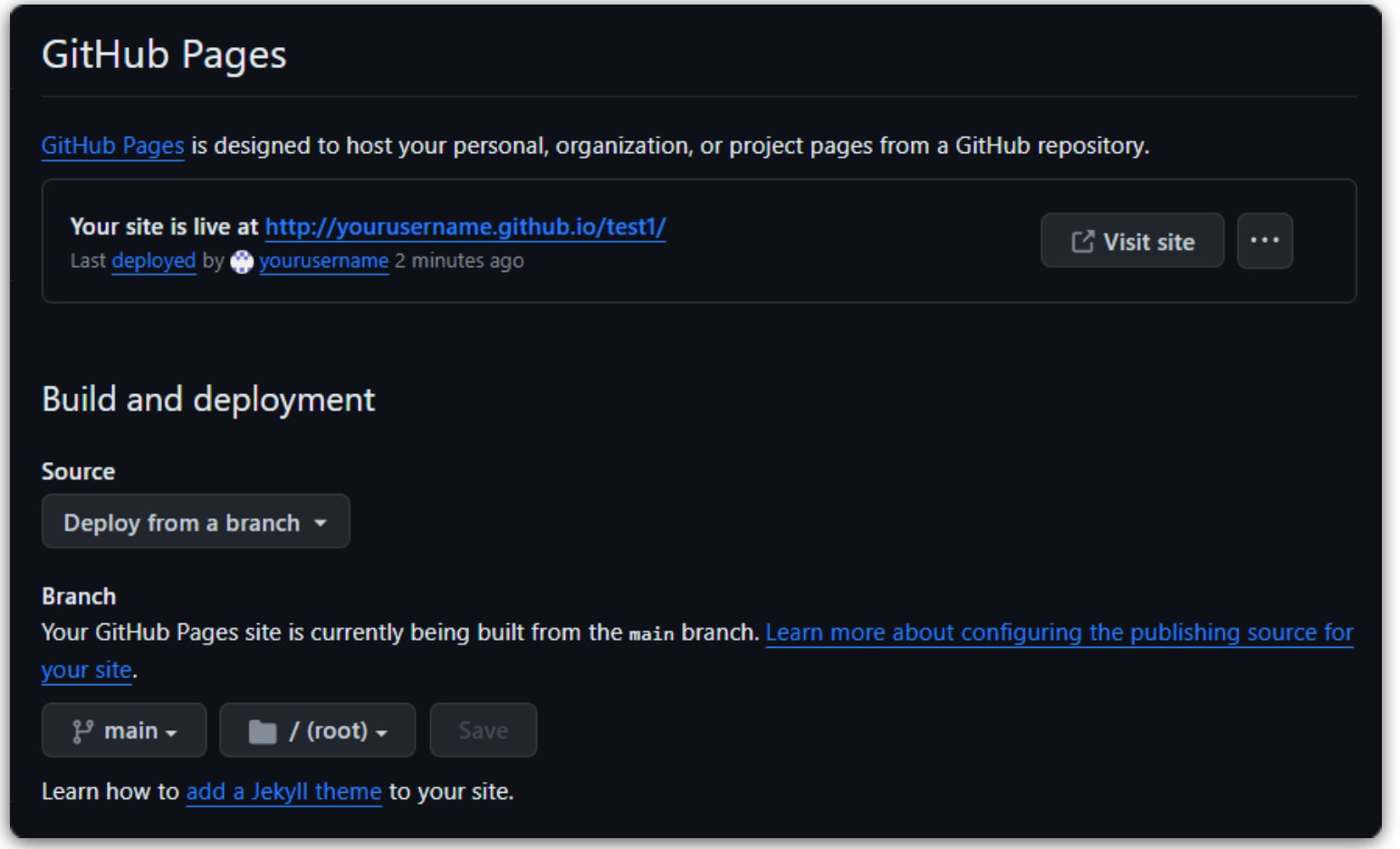


- And click on "Save".



Step 11

- Wait for 30s to 1 minute and reload the webpage on your browser.
  - You should see your hosted website url above the branch section!



That's it!

- Your game has been hosted in github pages, you can share the link to your friends, and it can be played in full screen.

Questions?