# Introduction to Generative Drawing

## *A Self-Paced Workbook*

*Generative Drawing is about creating images by following a set of simple rules, either with or without a computer!*

Prepared by Adam Herst
www.adamherst.art
adamherst@adamherst.com



2020-02-07

## Table of Contents

What's This Workbook About?....1
What is Generative Drawing?....2
Generative Drawing Without a Computer....4
Translating Our Drawing into Code....5
Generative Drawing with a Computer....6
Getting to Know the p5.js Web Editor....7
Functions and Variables....8
Playing a Sketch....9
Printing a Message to the Console Pane....10
Creating a Canvas....11
Changing the Colour of the Canvas....12
Drawing a Point....13
Drawing a Line....14
Drawing a Triangle....15
Drawing a Rectangle....16
Drawing an Ellipse....17
Setting Colours....18
Introducing Variability into the Sketches....19
Varying the Placement of Your Circle....20
Controlling the Draw Loop....21
Varying the Placement of Your Square....22
Placing Shapes Anywhere on the Canvas....23
Varying the Placement and Shape of Your Triangle....24
Preserving the Shape of Your Triangle....25
Varying the Size of Your Circle....26
Varying the Size of Your Square....27
Varying the Colour of Your Shapes....28
Creating a Custom Shape....30
Looping Through the Variations....31
Preparing Your Sketch to Share....32
Sharing Your Sketch....33
What's Next?....34
The Completed Sketch....35

## What's This Workbook About?

This workbook is about art first and computer programming second:

- If you like to draw, you should enjoy the activities in this workbook
- If you don't know how to write computer programs, don't worry; you don't need any previous programming experience to complete the activities in this workbook

### We are going to ...

- Look at some examples of generative drawing, created both with and without a computer
- Create a generative drawing without using a computer
- Use the web editor available for p5.js, a programming language developed for artists, to write computer programs
- Translate our generative drawings into p5.js programs
- Introduce randomization into the programs to produce a different sketch each time the program is run
- Share our sketches over the web

### We aren't going to ...

- Learn the intricacies of programming (but we are going to learn about a few very basic programming concepts)
- Do a lot of math (but we are going to do a little bit)

### Activities and Bonus Activities

This workbook is activity based. Each of the activities builds on the previous activity. At the end of the workbook, if you complete all the activities, you will have a working p5.js program. A version of the program is presented at the end of this booklet.

Bonus activities are extra. You don't have to complete them. They introduce more advanced concepts and point out some additional capabilities of p5.js.



## What is Generative Drawing?

Generative drawing is one type of generative art:

> *Generative art refers to art that in whole or in part has been created with the use of an autonomous system. An autonomous system in this context is generally one that is non-human and can independently determine features of an artwork that would otherwise require decisions made directly by the artist.*
>
> https://en.wikipedia.org/wiki/Generative_art

We're going to create drawings using two autonomous systems:

- a written set of rules
- a computer program

The features of our artwork that the autonomous systems will determine are:

- location
- size
- colour

### Why Should We Use a Computer to Create Drawings?

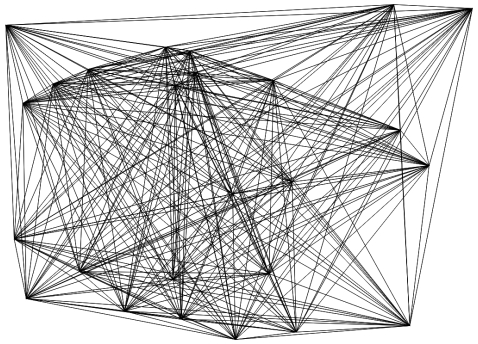
> *After all, if your program can't do anything more than what you could do quicker, better, or more creatively with a physical object, why bother using a computer at all.*
>
> https://www.futurelearn.com/courses/creative-coding/2/steps/35887

Generative drawing isn't new. Sol LeWitt's wall drawings are one example of drawing with code without a computer:

**On a wall surface, any**
**continuous stretch of wall,**
**using a hard pencil, place**
**fifty points at random.**
**The points should be evenly**
**distributed over the area**

**of the wall. All of the points should be connected by straight lines.**

https://observer.com/2012/10/here-are-the-instructions-for-sol-lewitts-1971-wall-drawing-for-the-school-of-the-mfa-boston



*At his core he was a minimalist, so much so that most of his famous pieces were not even executed by him in person. He did not sell paintings on canvases, instead he sold "codes" or procedures with specific instructions that would then be implemented by a draftsman, who was required to faithfully execute the instructions, but whose own hand and judgement led to the final formal outcome of the work.*

https://generativelandscapes.wordpress.com/2014/08/14/procedural-art-sol-lewitt-example-3-1

Vera Molnár began drawing with code using an "imaginary machine" and only later began using a physical computer.

### Activity: Sol LeWitt and Vera Molnár

Explore their work by:

1. Going to http://solvingsol.com/solutions/
2. Going to http://dada.compart-bremen.de/item/agent/14



## Generative Drawing Without a Computer

We'll use these materials to create a generative drawing without using a computer:

- one piece of paper
- one pencil
- three crayons of different colours

### Activity: Create a Generative Drawing

Create a generative drawing by:

1. Using the pencil, draw a triangle anywhere on the paper
2. Still using the pencil, draw a circle somewhere in the remaining space
3. Still using the pencil, draw a square somewhere in whatever space is left
4. Choose a crayon and colour in one of the shapes
5. Using one of the remaining crayons, colour in one of the remaining shapes
6. Using the third and last crayon, colour in the third and last shape

### Triangle, Square, Circle

*In 1923 Wassily Kandinsky circulated a questionnaire at the Bauhaus, asking respondents to fill in a triangle, square, and circle with the primary colors of red, yellow, and blue...He hoped to discover a universal correspondence between form and color, embodied in the equation red=square, yellow=triangle, blue=circle.*

https://bookofthrees.com/triangle-square-circle-a-psychological-test/

## Translating Our Drawing into Code

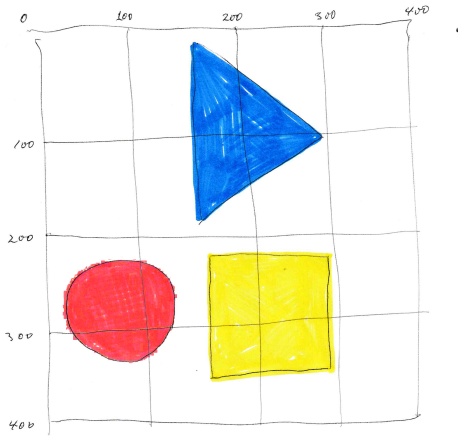
Now that we've created a generative drawing without using a computer, we're going to get it ready to translate into a p5.js program to create generative drawings using a computer.

### Activity: Prepare the Coordinate Grid

Using the pencil:

1. Draw a square around your three shapes, the smaller the better as long it contains all the shapes
2. Draw a line from the centre of the top of the square to the centre of the bottom, dividing the sketch into half
3. Draw a line from the centre of the left side of the square to the centre of the right side, dividing the sketch into four quarters
4. Draw lines dividing each of the quarters into four squares so that you have a grid of 16 squares

Your sketch should look something like this:






## Generative Drawing with a Computer

> *p5.js is a JavaScript library for creative coding, with a focus on making coding accessible and inclusive for artists, designers, educators, beginners, and anyone else!*
>
> https://p5js.org/

We can create sketches using p5.js in two ways:

- by downloading and installing the software
- by using the online p5.js web editor

For this workbook, we're going to use the web editor.

If you want to save your sketches or share your sketches via the p5.js editor website, you'll need to create an account:

- The only information you'll be asked to provide is your email address and a password.
- If you don't want to sign up, you can still use the editor but won't be able to save your work in progress or to share the sketch.

### Activity: Create a p5.js Web Editor Account

To create an account:

1. Go to https://editor.p5js.org/
2. In the top right corner, click on Sign up

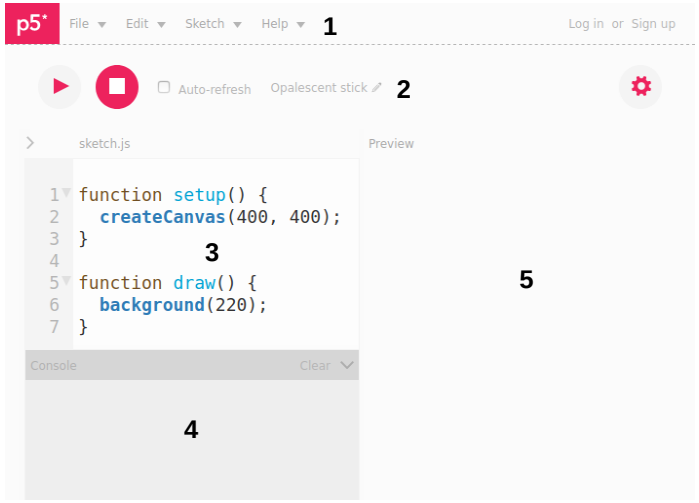
### Bonus Activity: Download and Install the p5.js Software

If you don't want to use the p5.js editor you can download the p5.js libraries and work locally with a code editor of your choice. Follow the instructions at:

- https://p5js.org/download/
- https://p5js.org/get-started/

## Getting to Know the p5.js Web Editor

We'll be using these parts of the web editor:




**1 - Menu Bar**

Used to save, open, share, and download sketches

**2 - Control Bar**

Contains the Start and Stop buttons used to control the sketch

**3 - Edit pane**

Used to type in the p5.js statements that make up a sketch

**4 - Console pane**

Used to display status and system messages

**5 - Preview pane**

Used to display the output of the sketch when it is played



## Functions and Variables

Functions and variables are basic programming concepts that we'll need to know about when we write programs in p5.js:

- A function is a block of organized, reusable code that is used to perform a single, related action.
- Variables are names given to computer memory locations which are used to store values in a computer program.

p5.js provides built-in functions and variables that we'll use in our sketches.

### setup() and draw()

All of our sketches will use the built-in functions setup() and draw():

- The setup() function is run once when the Start button is pressed.
- The draw() function will run over and over until the Stop button is pressed.

**In general, function names are followed by parentheses. Variables names are never followed by parentheses.**

Later in the workbook we will learn how to define our own functions and variables.

### Activity: Use the p5.js Online Reference

The p5.js website has an online reference to the built-in functions and variables. Look up the explanations for setup() and draw():

- Go to https://p5js.org/reference
- Enter the name of the variable or function you want to look up in the search field at the top right of the screen

**You might want to keep a browser window open to the p5.js reference during the workbook.**

## Playing a Sketch

We can play the sketch in the Edit pane by clicking on the Start button. The sketch will be drawn in the Preview pane. We can stop the sketch by clicking on the Stop button.

### Typing in p5.js Statements

The Edit pane is used to type in the p5.js statements that will make up the sketch.

**p5.js statements end with a semicolon.**

The Edit pane is pre-filled with a skeleton of a p5.js sketch:

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

function draw() {
  background(220);
}
```

### Starting and Stopping the Sketch

The Start button starts the sketch. The Stop button stops the sketch. They are located on the Control Bar just above the Edit pane.

### Displaying the Sketch

The Preview pane is located to the right of the Edit pane. It shows the result of playing the sketch.

### Activity: Play the Skeleton Sketch

Play the skeleton sketch by:

1. Pressing the Start button to start the sketch. The sketch should draw a gray square in the Preview pane.
2. Pressing the Stop button to stop the sketch. The gray square should disappear.



## Printing a Message to the Console Pane

If a sketch can't be played, messages about possible reasons why are shown in the Console pane. We can print our own messages to the Console pane using the p5.js built-in function:

- print()

### print()

The print() function writes to the console area of your browser. This function is often helpful for looking at the data a program is producing.

### Activity: Print a Message to the Console pane

Print a message to the Console pane by:

1. In the Editor pane, after the createCanvas() statement, add the statement:

   ```
   print('What colour is your circle?');
   ```

   **Make sure to include the quotation marks around the string.**

2. Press the Start button. The message is displayed in the Console pane.

### Comments

If a p5.js statement contains the characters //, any characters in the statement after the // are ignored. Comments are useful to explain what statements in a sketch do. The sample sketch at the end of the booklet uses comments to show where an activity comes from.

### Bonus Activity: Print a Message Multiple Times

Statements in the setup() function are executed once. Statements in the draw() function are executed over and over until the Stop button is pressed. You can print a message multiple times by:

1. After the background() statement, add the statement:

   ```
   print('Message number: ' + frameCount);
   ```

2. Press the Stop button to stop the sketch.

## Creating a Canvas

Before we can draw a sketch we need to create a canvas to draw on. We create a canvas with the built-in function:

- createCanvas()

### createCanvas()

The createCanvas() function creates a canvas element in the document, and sets the width and height dimensions of it in pixels:

```
createCanvas(width, height);
```

**createCanvas() should be called only once at the start of setup.**

The skeleton setup() creates a default canvas to draw on:

```
createCanvas(400, 400);
```

- the first 400 represents the width of the canvas in pixels
- the second 400 represents the height of the canvas in pixels

### Activity: Change the Size of the Canvas

Change the size of the canvas by using different values for the width and height. Make the canvas 200 pixels wide and 600 pixels high by:

1. Change the createCanvas() statement to:
   ```
   createCanvas(200, 600);
   ```
2. Click on the Start button to draw the new canvas.
3. Click on the Stop button to stop the sketch
4. Change the createCanvas() statement back to:
   ```
   createCanvas(400, 400);
   ```

**Make sure to change the canvas width and height back to the original dimensions. We're going to need a 400x400 pixel canvas for the activities that follow.**



## Changing the Colour of the Canvas

The default background is transparent. We can change the colour with the built-in function:

- background()

### background()

The background() function sets the color used for the background of the p5.js canvas.

```
background(colour)
```

- background() is typically used within draw() to clear the display window at the beginning of each frame
- It can be used inside setup() to set the background on the first frame of animation or if the background need only be set once.

### CSS colour strings

The background() function in the skeleton sketch sets the colour using a numeric value (220 represents a shade of grey):

```
background(220);
```

An easier way to specify a colour in p5.js is to use a CSS colour string, for example:

```
'lightgrey'
```

**Make sure to include the quotation marks around the string.**

### Activity: Set the Background Colour Using a CSS String

Change the background to light grey by:

1. Change the background() statement to:
   ```
   background('lightgrey');
   ```

### Bonus Activity: Set the Background Using HSB Values

You can also use HSB values to specify a colour.

1. Explore colorMode() in the p5.js online reference. Use it to set the background to light grey using HSB values.

## Drawing a Point

p5.js has built-in functions to draw points and to change the width of the stroke that draws a point, line, or the border around shapes:

- point()
- strokeWeight()

### point()

The point() function draws a single pixel at the specified coordinate on the canvas:

```
point(x, y);
```

X and Y are numbers that identify a location on the canvas:

- X is the horizontal distance in number of pixels from the top left corner
- Y is the vertical distance in number of pixels from the top left corner

**The top left corner of the canvas has the coordinates (0,0).**

### strokeWeight()

By default, points, lines, and the borders around shapes are only one pixel in size. To make them bigger, use the strokeWeight() function to set the width of the stroke:

```
strokeWeight(width);
```

- width is the number of pixels to use

### Activity: Draw a Point Ten Pixels in Size

Set the size of the point to 10 pixels and draw it by:

1. In setup(), after the print() statement, add the statement:
   ```
   strokeWeight(10);
   ```
2. In draw(), after the background() statement, add the statement:
   ```
   point(200,200);
   ```



## Drawing a Line

p5.js has a built-in function to draw a line:

- line()

### line()

The line() function draws a line, a direct path between two points.

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

- x1 and y1 are the coordinates of the point to start the line
- x2 and y2 are the coordinates of the point to end the line

### Activity: Draw a Grid on Your Sketch

Draw a grid with 16 squares on your sketch:

1. Draw a line from the centre of the top of the canvas to the centre of the bottom to divide the sketch into half. After the point() statement add the statement:
   ```
   line(200, 0, 200, 400);
   ```
2. Draw a line from the centre of the left side of the canvas to the centre of the right side to divide the sketch into four squares. Add the statement:
   ```
   line(0, 200, 400, 200);
   ```
3. Draw lines dividing each of the quadrants into four squares so that you have a grid of 16 squares. Add the vertical lines by adding the statements:
   ```
   line(100, 0, 100, 400);
   line(300, 0, 300, 400);
   ```
   Add the horizontal lines by adding the statements:
   ```
   line(0, 100, 400, 100);
   line(0, 300, 400, 300);
   ```

## Drawing a Triangle

p5.js has a built-in function to draw a triangle:

- triangle()

### triangle()

The triangle() function creates a shape by connecting three points with straight lines:

```
triangle(x1, y1, x2, y2, x3, y3)
```

- x1 and y1 represent the coordinates of the first point
- x2 and y2 represent the coordinates of the second point
- x3 and y3 represent the coordinates of the third point

### Activity: Draw your Triangle

Draw the triangle from your paper drawing by:

1. Using the grid we drew on the paper, estimate the coordinates of the 3 corners of the triangle.

**Use the same coordinate values for the grid on the paper as we used for the grid we just drew on our p5.js canvas: (0, 0) for top left corner, (400, 400) for the bottom right.**

2. After the last line() statement, add the statement:

```
triangle(0, 0, 0, 200, 200, 200);
```

**Use the coordinate values from your drawing for the triangle() statement.**



## Drawing a Rectangle

p5.js has a built-in function to draw a rectangle:

- rect()

### rect()

The rect() function draws a rectangle:

```
rect(x, y, width, height);
```

- x and y represent the coordinates of the upper-left corner of the rectangle
- width is the width of the rectangle
- height is the height of the rectangle

**A rectangle with a height equal to its width is a square.**

### Activity: Draw Your Square

Draw the square from your paper drawing:

1. Using the grid we drew on the paper, estimate the coordinates of the upper-left corner
2. Using the grid we drew on the paper, estimate the width of the square. Use that value for the height as well.
3. After the triangle() statement, add the statement:

```
rect(200, 0, 200, 200);
```

**Use the coordinate values from your drawing for the rect() statement.**

**Use the width for both the width and height parameters.**

### Bonus activity: Use a different rectMode()

The way the parameters to rect() are interpreted may be changed with the rectMode() function:

1. Explore the rectMode() function in the online reference
2. Draw your rectangle using CENTER mode.

## Drawing an Ellipse

p5.js has a built-in function to draw an ellipse:

- ellipse()

### ellipse()

The ellipse() function draws an ellipse:

```
ellipse(x, y, width, [height])
```

- x and y represent the coordinates of the centre of the ellipse
- width is the width of the ellipse
- height is the height of the ellipse. If no height is specified, the value of width is used for both the width and height.

**An ellipse with a height equal to its width is a circle.**

### Activity: Draw Your Circle

Draw the circle from your paper drawing:

1. Using the grid we drew on the paper, estimate the values for the X and Y coordinates of the centre of the circle.
2. Using the grid we drew on the paper, estimate the width of the circle by:
    1. Drawing a horizontal line through the centre of the circle
    2. Subtracting the X coordinate of the left end of the line from the X coordinate of the right end.
3. After the rect() statement, add the statement:

    ```
    ellipse(200, 300, 200);
    ```

**Use the coordinate and width values from your drawing for the ellipse() statement.**



## Setting Colours

p5.js has built-in functions to set the colours of lines and the inside of shapes and to turn off drawing borders around or filling shapes:

- stroke() and noStroke
- fill() and noFill()

### stroke() and fill()

The stroke() function sets the color used to draw lines and borders around shapes. The fill() function sets the color used to fill shapes:

```
stroke(colour)
fill(colour);
```

- colour can be a CSS string

**All shapes drawn after the stroke() and fill() statements will use the specified colours.**

### noStroke() and noFill()

The noStroke() function disables drawing the outline around shapes. The noFill() function disables filling shapes with colours and makes them transparent.

```
noFill();
noStroke();
```

### Activity: Colour Your Shapes

Colour your shapes by:

1. Turn off the borders around the shapes. After the background() statement, add the statement:

    ```
    noStroke();
    ```

2. Before **each** of the triangle(), rect(), and ellipse() statements, add the statement:

    ```
    fill('red');
    ```

**Replace 'red' with the string for the colour of the shape from your drawing.**

## Introducing Variability into the Sketches

p5.js has a built-in function to generate random numbers we can use to vary the placement of shapes each time the program is run:

- random()

### random()

The random() function returns a random floating-point number:

```
random([min],[max])
```

- if no argument is given, returns a random number from 0 up to (but not including) 1
- if one argument is given, returns a random number from 0 up to (but not including) the number
- if two arguments are given, returns a random number from the first argument up to (but not including) the second argument

### Activity: Print Random Numbers to the Console

Print random numbers to the console by:

1. After the last print() statement, add the following statements:

```
print(random());
print(random(10));
print(random(90,100));
```

1. Run the sketch. Random numbers are printed to the Console pane
2. Run the sketch again. Different numbers are printed.

### Bonus Activity: Convert the Numbers to Integers

You can convert the floating-point numbers returned by random() into integers using the int() function:

1. Explore the int() function in the online reference. Print random integers to the Console pane.



## Varying the Placement of Your Circle

We can use the random() function to generate random numbers that we can pass as the parameters to the ellipse() function to use as the X and Y coordinates of our circle. First, we're going to save the values of the coordinates from our drawings in variables using:

- let

### let

The let keyword is used to specify a name for our variable, so we can refer to it later, and to optionally assign a value to it:

```
let myVariable = 1;
```

We're going to declare our variables at the very top of the sketch, outside of the setup(), draw(), or any other functions. This will let every other part of the sketch use them.

**Where variables are declared determines which other parts of our sketch will be able to use them.**

### Activity: Randomize the Placement of Your Circle

Save the initial values of the X and Y coordinates in variables and then generate new values for them each time the sketch is run by:

1. Before the setup() statement, add the statements:

```
let circleX = 200;
let circleY = 300;
```

**Use the X and Y coordinate values of your circle**

2. Change the ellipse() statement to be:

```
ellipse(random(circleX),random(circleY), 200);
```

### Bonus Activity: Make the Circle Follow Your Mouse

p5.js has two built-in variables that always hold the current position of the mouse (mouseX and mouseY):

1. Use mouseX and mouseY as the parameters for X and Y in the ellipse() statement to make the circle follow your mouse pointer.

## Controlling the Draw Loop

The setup() function runs once. After that, the draw() function keeps running over and over until you press the Stop button. p5.js provides built-in functions we can use to control the draw() loop:

- noLoop() and loop()

### noLoop() and loop()

The noLoop() function stops p5.js from continuously executing the code within draw(). If the loop() function is called, the code in draw() begins to run continuously again.

### Activity: Stop the Circle from Bouncing Around

We can stop the circle from bouncing around using the noLoop() function:

1. As the last line of the setup() function, add the statement:

   ```
   noLoop();
   ```

2. Click the Play button. Click the Play button again to run the sketch again or click the Stop button to stop it.

### Bonus activity: Click the Mouse to Make the Circle Move

The mouseClicked() function is called once after a mouse button has been pressed and then released. By default, this function does nothing. To be useful to us, we have to declare the function and add the code we want to be executed:

The redraw() function executes the code within draw() one time. This function allows the sketch to update the display window only when necessary.

To redraw the sketch whenever the mouse is clicked on it:

1. At the very bottom of the sketch, after the draw() function, add these statements:

   ```
   function mouseClicked() {
     redraw();
   }
   ```



## Varying the Placement of Your Square

We can use the random() function to generate random numbers that we can pass as the parameters to rect() as the X and Y coordinates to the function that draws our square:

```
rect(x, y, width, height)
```

### Activity: Randomize the Placement of Your Square

We can vary the placement of the square by saving our initial values in variables and then generating new values each time the sketch is run using `random()`:

1. After the last let statement at the top of the sketch add the statements:

   ```
   let squareX = 200;
   let squareY = 0;
   ```

2. Change the rect() statement to be:

   ```
   rect(random(squareX),random(squareY), 200, 200);
   ```

## Placing Shapes Anywhere on the Canvas

Using the variables we defined for the X and Y coordinates of the centre of the circle (circleX and circleY) and for the top left corner of the square (squareX and squareY) as the parameters to random() means that the centre of our circle can only be between the top left corner of the canvas (0, 0) and the point with coordinates (circleX, circleY) and the top left corner of our square can only be between (0, 0) and (squareX, squareY).

p5.js has built-in variables that contain values for the width and height of the canvas:

- width
- height

### width

width is set to the width of the drawing canvas in pixels. This value is set by the first parameter to the createCanvas() function.

### height

height is set to the height of the drawing canvas in pixels. This value is set by the second parameter to the createCanvas() function.

**We don't have to declare built-in variables with let before we use them.**

### Activity: Replace the Parameters to random()

We can use the built-in variables width and height to vary the placement of the circle and square to be anywhere on the canvas by:

1. Change the ellipse() statement to be:

```
ellipse(random(width), random(height), 200);
```

2. Change the rect() statement to be:

```
rect(random(width),random(height), 200, 200);
```



## Varying the Placement and Shape of Your Triangle

We can use the random() function to generate random numbers that we can pass as the parameters for the X and Y coordinates to the function that draws our triangle:

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

Unlike the ellipse() and rect() functions, which take X and Y coordinates for a single point to place the shape, the triangle function takes three pairs of X and Y coordinates, one for each point of the triangle.

If we use random() for each of the six coordinates, the triangle won't just be placed in a different location, it will be drawn with a different size and shape.

### Activity: Randomize the Coordinates of Your Triangle

Change the shape and placement of the triangle by:

1. After the last let statement at the top of the sketch add the statements:

```
let triangleX1 = 0;
let triangleY1 = 0;
let triangleX2 = 0;
let triangleY2 = 200;
let triangleX3 = 200;
let triangleY3 = 200;
```

2. Change the triangle() statement to be:

```
triangle(random(triangleX1), random(triangleY1),
random(triangleX2), random(triangleY2),
random(triangleX3), random(triangleY3));
```

## Preserving the Shape of Your Triangle

We can preserve the shape of the triangle by picking one corner and then expressing the X and Y coordinates of the other two corners in relation to the X and Y coordinates of the first corner.

### Activity: A Little Bit of Math

Express the X and Y coordinates of the second and third corners as the difference from the X and Y coordinates of the first corner by:

1. At the top of the sketch, change the let statement for triangleX2:

   ```
   let triangleX2 = 0 - triangleX1;
   ```

2. Change the let statement for triangleY2:

   ```
   let triangleY2 = 200 - triangleY1;
   ```

3. Change the let statement for triangleX3:

   ```
   let triangleX3 = 200 - triangleX1;
   ```

4. Change the let statement for triangleY3:

   ```
   let triangleY3 = 200 - triangleY1;
   ```

5. Generate random numbers to use as the X and Y coordinates for the first corner. Before the triangle() statement, add the lines:

   ```
   triangleX1 = random(width);
   triangleY1 = random(height);
   ```

6. Draw the triangle expressing the X and Y coordinates of the second and third corners as a distance from the X and Y coordinates of the first corner. Change the triangle() statement to be:

   ```
   triangle(triangleX1, triangleY1, triangleX1 +
   triangleX2, triangleY1 + triangleY2, triangleX1
   + triangleX3, triangleY1 + triangleY3);
   ```



## Varying the Size of Your Circle

We were able to vary the placement of shapes by generating random numbers to use as the parameters for the X and Y coordinates. We can vary the size of the circle by using random() to generate a random number to use as the parameter for its width.

### Activity: Vary the Width of Your Circle

To vary the size of the circle, declare a variable for the width of the circle, assign the values from your drawing to use as the initial value, and then use that initial value to generate a random number to use as the new width:

1. After the let statement for circleY, add the statement:

   ```
   let circleWidth = 200;
   ```

2. Change the ellipse() statement to be:

   ```
   ellipse(random(width), random(height),
   random(circleWidth));
   ```

### Activity: Let the Circle be as Wide as the Canvas

Using the circleWidth variable we defined as the parameter to random() means that the circle can only be smaller than the original. We can use the built-in width variable as the parameter to random() to allow the circle to be as wide as the canvas:

1. Change the ellipse() statement to be:

   ```
   ellipse(random(width), random(height),
   random(width));
   ```

### Bonus Activity: Keep the Circle Smaller than the Canvas

To keep the circle smaller than the canvas, it's width has to be smaller than whichever of the width or height is smallest:

1. Explore the if-else statement in the online reference. Use it to pass the smallest of width or height to random().

## Varying the Size of Your Square

We can vary the size of the square by using random() to generate random number to use as the parameter for its width and height.

### Activity: Vary the Size of Your Square

To vary the size of the square, declare variables for its width and height, assign the values from your drawing to use as the initial values, and then use those initial values to generate random numbers to use as the width and height:

1. After the let statement for squareY, declare the new variables and assign their initial values:

```
let squareWidth = 200;
let squareHeight = 200;
```

2. Change the rect() statement to be:

```
rect(random(width), random(height),
random(squareWidth), random(squareHeight));
```

### Activity: Preserve the Proportions of Your Square

Unlike the ellipse() statement, which lets us omit the height parameter, the rect() statement needs parameters for both the width and the height. Because we are using two different random numbers for the width and height, the rect() statement (mostly) draws a rectangle, not a square.

To preserve the proportions of the square, we have to generate and store a random number and then use it as the value for both the width and the height parameters:

1. Before the rect() statement, add the statements:

```
squareWidth = random(width);
squareHeight = squareWidth;
```

2. Change the rect() statement to be:

```
rect(random(width), random(height),squareWidth,
squareHeight);
```



## Varying the Colour of Your Shapes

We're going to use a new variable type, an array, as a way to vary the colour of the shapes each time the sketch is played. We can change the order of the elements in an array using the built-in function:

- shuffle()

### Array

An Array is a special variable type which can hold more than one value at a time. Up to now our variables could only hold one value at a time.

We could declare three variables and assign CSS strings as their value with these statements:

```
let colour0 = "red";
let colour1 = "yellow";
let colour2 = "blue";
```

Instead of declaring three variables, we can declare an array variable named arrayOfColours with this statement:

```
let arrayOfColours = [ 'red', 'yellow', 'blue' ];
```

- The left and right brackets indicate that this is an array variable.
- Each element in the array is separated by a comma.
- Each element in the array is identified by an index number starting with 0

We can reference the value stored in a specific element of the array by using the name of the array followed by the index number of that element. To reference the value 'red', we would use:

```
arrayOfColours[0]
```

**Array index numbers start at 0.**

### shuffle()

The shuffle() function randomizes the order of the elements of an array passed to it as a parameter:

```
shuffle(array, boolean);
```

- array is the name of the array variable
- boolean is one of true or false. true means to modify the array passed as the parameter. false means to return a new array.

**We want to use true for this workbook.**

### Activity: Shuffle the Colours of the Shapes

Replace the CSS strings we've been using as parameters to the fill() function with elements from an array holding the CSS strings by:

1. After the last let statement, add the statement:

```
let arrayOfColours = [ 'red', 'yellow',
'blue' ];
```

**Use the CSS strings for the colours from your drawing as the values for the array elements.**

2. After the background() statement, add the statement:

```
shuffle(arrayOfColours, true);
```

3. Change the first fill() statement to be:

```
fill(arrayOfColours[0]);
```

4. Change the second fill() statement to be:

```
fill(arrayOfColours[1]);
```

5. Change the third fill() statement to be:

```
fill(arrayOfColours[2]);
```

**Each fill() statement uses a different index number to reference different elements in the array variable arrayOfColours.**



## Creating a Custom Shape

We can create our own shapes using the built-in functions:

- beginShape()
- endShape()
- vertex()

### beginShape() and endShape()

The beginShape() and endShape() functions allow the creation of more complex forms:

```
beginShape();
      // add any number of vertex() statements
endShape();
```

- beginShape() begins recording vertices for a shape
- endShape() stops recording vertices

### vertex()

The vertex() function specifies a point of your shape to be connected by a line:

```
vertex(x, y);
```

### Activity: Add a Five-sided Shape to Your Sketch

You can create a shape with any number of vertex() statements. Add a five-sided shape and colour the shape using one of the three available colours choosen at random by:

1. After the ellipse() statement, add the statements:

```
fill(arrayOfColours[int(random(0,3))]);
beginShape();
vertex(random(width), random(height));
vertex(random(width), random(height));
vertex(random(width), random(height));
vertex(random(width), random(height));
vertex(random(width), random(height));
endShape();
```

## Looping Through the Variations

By default, p5.js loops continuously through the draw() function, executing the code within it. Earlier, we turned off looping with the noLoop() function. We can resume looping through variations of our sketch and control the speed of the loop with the built-in functions:

- loop()
- frameRate()

### loop()

The loop() function resumes looping through the draw function if it has been turned off by a call to the noLoop() function:

```
loop();
```

### frameRate()

The frameRate() function specifies the number of frames (or loops) to be displayed every second. A frame rate of 24 frames per second (usual for movies) or above will be enough for smooth animations:

```
frameRate(24);
```

### Activity: Loop Through One Sketch Each Second

Loop through one sketch each second by turning looping back on and specifying a frame rate of 1:

1. After the noLoop() statement, add the statements:

```
loop();
frameRate(1);
```

### Bonus Activity: Frame Rates of Less than 1

Frame rates of less than 1 can be specified using a floating-point number between 0 and 1. Make your sketch loop through one sketch every 10 seconds.



## Preparing Your Sketch to Share

Before we share our sketches, we can make the canvas as large as the browser window and change the background colour to something more interesting than grey using:

- background()
- windowWidth
- windowHeight

### background()

The background() function sets the colour used for the background of the p5.js canvas. This function is typically used within draw() to clear the display window at the beginning of each frame.

As we've seen, one of the easiest way to specify a colour for the background is by using a CSS string:

```
background('white');
```

### windowWidth and windowHeight

windowWidth and windowHeight are built-in variables that hold the values in pixels of the browser window's width and height:

**Because windowWidth and windowHeight are built-in variables, we do not have to declare them with let before we use them**

### Activity: Change the Colour and Size of the Background

Change the background colour to white or black and make it as large as the browser window:

1. Change the background() statement to be:

```
background('black');
```

2. Change the createCanvas() statement to be:

```
createCanvas(windowWidth, windowHeight);
```

## Sharing Your Sketch

We can share our sketches directly from the p5.js web editor website or by downloading the sketch and hosting it on our computer or web server.

### Sharing from the Website

Sketches can be shared from the website in a number of ways:

- Embed
- Present
- Fullscreen
- Edit

To share a sketch, go to the File menu and select Share.

### Downloading to a Computer

You can download a .zip file of the files that make up a sketch and host them from your own computer or web server. The .zip file will contain these files:

- index.html
- sketch.js
- style.css
- p5.js
- p5.sound.min.js

To download a sketch, go to the File menu and select Download.

### Activity: Share Your Sketch

Go to the File menu and select Share. URLs for each of the sharing methods will be displayed.

1. Copy and paste the URLs for Present, Fullscreen, and Edit into a browser window to see how they differ

### Bonus Activity: Download Your Sketch

To download your sketch, go to the File menu and select Download. Download and unzip the file. Use your browser to find and open the index.html file.



## What's Next?

Try these resources to learn more about what p5.js can do and how you can use it.

### Resources on the p5.js Web Site

These are just a few of the links at https://p5js.org/learn/:

- p5.js Overview: An overview of the main features of p5.js.
- Coordinate System and Shapes: Drawing simple shapes and using the coordinate system.
- Interactivity: Introduction to interactivity with the mouse and keyboard.
- Program Flow: Introduction to controlling program flow in p5.js.
- Color: An introduction to digital color.
- Beyond the canvas: Creating and manipulating elements on the page beyond the canvas.
- 3D/WebGL: Developing advanced graphics applications in p5.js using WEBGL mode.

### Online Courses

There are free courses offered by some of the online learning websites:

- Creative Coding on edX

  https://www.edx.org/course/creative-coding

- Introduction to Programming for the Visual Arts with p5.js on Kadenze

  https://www.kadenze.com/courses/introduction-to-programming-for-the-visual-arts-with-p5-js-iii

## The Completed Sketch

Here's a copy of the sketch created by completing the activities in this workbook. Statements from every activity are included and commented out as they were changed. Comments at the end of each line refer back to the section the activity came from.

```
let circleX = 200; // Varying the Placement of the Circle
let circleY = 300; // Varying the Placement of the Circle
let circleWidth = 200; //Varying the Size of Your Circle

let squareX = 200; // Varying the Placement of the Square
let squareY = 0; // Varying the Placement of the Square
let squareWidth = 200; // Varying the Size of Your Square
let squareHeight = 200; // Varying the Size of Your Square

let triangleX1 = 0; // Varying the Placement and Shape of Your
Triangle
let triangleY1 = 0; // Varying the Placement and Shape of Your
Triangle
// let triangleX2 = 0; // Varying the Placement and Shape of
Your Triangle
// let triangleY2 = 200; // Varying the Placement and Shape of
Your Triangle
// let triangleX3 = 200; // Varying the Placement and Shape of
Your Triangle
// let triangleY3 = 200; // Varying the Placement and Shape of
Your Triangle
let triangleX2 = 0 - triangleX1; // Preserving the Shape of
Your Triangle
let triangleY2 = 200 - triangleY1; // Preserving the Shape of
Your Triangle
let triangleX3 = 200 - triangleX1; // Preserving the Shape of
Your Triangle
let triangleY3 = 200 - triangleY1; // Preserving the Shape of
Your Triangle

let arrayOfColours = ['red', 'yellow', 'blue']; // Varying the
Colour of the Shapes


function setup() {
  //  createCanvas(400, 400); // Skeleton sketch
  // createCanvas(200,600); // Creating a Canvas
  createCanvas(windowWidth, windowHeight); // Preparing our
Sketches to Share

  print('What colour is your circle?'); // Printing a Message
to the Console Pane

  print(random()); // Introducing Variability into the Sketches
```



```
  print(random(10)); // Introducing Variability into the
Sketches
  print(random(90, 100)); // Introducing Variability into the
Sketches

  strokeWeight(10); // Drawing a Point

  // noLoop(); // Controlling the Draw Loop
  loop(); // Looping Through the Variations
  frameRate(1); // Looping Through the Variations
}

function draw() {
  // background(220); // Skeleton sketch
  //  background('lightgrey'); // Changing the Colour of the
Canvas
  background('black'); // Preparing our Sketches to Share

  shuffle(arrayOfColours, true); // Varying the Colour of the
Shapes

  noStroke(); // Setting Colours

  point(200, 200); // Drawing a Point

  line(200, 0, 200, 400); // Drawing a Line
  line(0, 200, 400, 200); // Drawing a Line
  line(100, 0, 100, 400); // Drawing a Line
  line(300, 0, 300, 400); // Drawing a Line
  line(0, 100, 400, 100); // Drawing a Line
  line(0, 300, 400, 300); // Drawing a Line

  // fill('red'); // Setting Colours
  fill(arrayOfColours[0]); // Varying the Colour of the Shapes

  // triangle(0, 0, 0, 200, 200, 200); // Drawing a Triangle
  // triangle(random(triangleX1), random(triangleY1),
random(triangleX2), random(triangleY2), random(triangleX3),
random(triangleY3)); // Varying the Placement and Shape of Your
Triangle
  triangleX1 = random(width); // Preserving the Shape of Your
Triangle
  triangleY1 = random(height); // Preserving the Shape of Your
Triangle
  triangle(triangleX1, triangleY1, triangleX1 + triangleX2,
triangleY1 + triangleY2, triangleX1 + triangleX3, triangleY1 +
triangleY3); // Preserving the Shape of Your Triangle

  // fill('yellow'); // Setting Colours
  fill(arrayOfColours[1]); // Varying the Colour of the Shapes


  // rect(200, 0, 200, 200); // Drawing a Rectangle
  // rect(random(squareX),random(squareY), 200, 200); //
Varying the Placement of the Square
```

```
  // rect(random(width), random(height), 200, 200); // Placing
Shapes Anywhere on the Canvas
  // rect(random(width), random(height), random(squareWidth),
random(squareHeight)); // Varying the Size of Your Square
  squareWidth = random(width); // Varying the Size of Your
Square
  squareHeight = squareWidth; // Varying the Size of Your
Square
  rect(random(width), random(height), squareWidth,
squareHeight); // Varying the Size of Your Square

  // fill('blue'); // Setting Colours
  fill(arrayOfColours[2]); // Varying the Colour of the Shapes


  // ellipse(200, 300, 200); // Drawing and Ellipse
  // ellipse(random(circleX),random(circleY), 200); // Varying
the Placement of the Circle
  // ellipse(random(width), random(height), 200); //Placing
Shapes Anywhere on the Canvas
  //  ellipse(random(width), random(height),
random(circleWidth)); // Varying the Size of Your Circle
  ellipse(random(width), random(height), random(width)); //
Varying the Size of Your Circle

  fill(arrayOfColours[int(random(0, 3))]); // Creating a Custom
Shape
  beginShape(); // Creating a Custom Shape
  vertex(random(width), random(height)); // Creating a Custom
Shape
  vertex(random(width), random(height)); // Creating a Custom
Shape
  vertex(random(width), random(height)); // Creating a Custom
Shape
  vertex(random(width), random(height)); // Creating a Custom
Shape
  vertex(random(width), random(height)); // Creating a Custom
Shape
  endShape(); // Creating a Custom Shape
}
```