Week 1 - Coordinates, shapes and colour

Outcomes

- Set up development environment
- Understand the principles of locating points on screen
- Drawing shapes
- Understanding additive colour and using RGB colour space
- Use browser-based tools for debugging and logging
- Animate shape or colour using variables

What is p5.js?

During this series of workshops you will be using a library called **p5.js** to learn the fundamentals of programming. The p5.js project is the most recent part of a complex history of open-source, creative coding libraries going back to the early 2000s. It is supported by the **Processing Foundation**, which is a not-for-profit organisation that emerged from the creative coding library **Processing**.

From a technical perspective, p5.js is simply a JavaScript library. A library is a collection of code put together to simplify a task or a collection of tasks. In this case p5.js provides a lot of functionality that makes it easy to draw shapes, colours and handle user interaction within a web page.

This video from Daniel Shiffman is a good introduction to p5.js and the creative coding platforms that preceded it:

https://player.vimeo.com/video/137979313

Supporting code

The code for this workshop is hosted on Github, which is a web-based repository for hosting and versioning code.

Download the code and unzip it on your desktop.

> The code is also available to **view directly on Github's website**.

A p5.js project

In this exercise you will set up a p5.js project using the Atom text editor, then examine the different files and run the code in a browser. The code to support this section is located in the following directory and is available to view on **Github**:

```
/00_empty_project/
```

Topics

- p5.js project structure
- What is a sketch?
- What do the `setup()` and `draw()` functions do?
- Adding your project to Atom
- Using the browser debugging tools

A p5.js project

Below is the structure of a p5.js project, which is essentially a web project made up of HTML and JavaScript files.

```
/00_empty_project/
  ├── index.html
  ├── libraries
  │   └── p5
  │       └── p5.min.js
  └── sketch.js
```

index.html

p5.js is a JavaScript library designed for drawing to a web page. For JavaScript code to run in a browser it needs to be included in a HTML file. The `index.html` file is the 'entry point' for the browser to access our project code. Note the use of the `<script>` tag to import two JavaScript files (line 7 & 8).

The first JavaScript file (`libraries/p5/p5.min.js`) is the p5.js library containing a vast amount of code that we can use without having to fully understand.

The second JavaScript file (`sketch.js`) is where we write our own code.

sketch.js

Below is the minimum required code for a p5.js sketch. This is simply an empty template for us to start coding and will not produce any visual results.

To summarise, we now know that when the browser loads the `index.html` file, it will import the p5.js library and the `sketch.js` file, and then execute the code we have written.

> Sketch - Why a sketch?

`setup()` and `draw()`

Within the sketch we have two functions: `setup()` and `draw()`. p5.js calls/runs these functions for us in a particular order. The setup function runs first and only once. The draw function then runs repeatedly until the web page is closed.

sketch-setup-draw-01.png

Debugging

When coding in any language and with any level of experience or expertise, you will almost always encounter bugs. Writing code is often a trial and error process. Therefore, to be productive programmers we need *debug* our code in order to identify and fix problems. This means using tools to show us where errors in our code occur whilst it is being executed in its runtime environment.

p5.js is written in JavaScript and therefore the environment for running our code will be the browser. There are developer tools built into all the major browsers that van be used for debugging. For now, we recommend using Chrome so we are all using the same tools throughout the workshop. Chrome has an easy to use and fully featured set of developer tools also known as DevTools.

- Take a look at Chrome's **instructions on how to use the DevTools**, in particular the **Accessing the DevTools section**
- A **more involved introduction to developer tools** from HTML5Rocks.
- p5.js has a very good **Field Guide to Debugging**. It explains that debugging is a creative problem solving task and stresses the importance of taking time to observing the problem in order to understand it.

Exercise

- Add the `00_empty_project` directory to Atom
- Open `index.html` in a browser
- Use the developer tools to see logged messages

Coordinates and Shapes

In this exercise you will learn how to locate and target positions (i.e. pixels) on screen for drawing. We will also learn how to use some basic functions of p5.js for making primitive shapes. The code to support this section is located in the following directory and is available to view on **Github**:

```
/01_coordinates_and_shapes/
```

Topics

- Comments
- Using p5.js functions
- Drawing some simple shapes
- Locating points on the screen using Cartesian Coordinates

Comments

When writing code it is a good idea to sometimes write notes to yourself or other coders to explain what the code is meant to be doing. The way we do this is by adding *comments*. Comments can be added in two ways:

1. Single line comment
   Using the double forward slash (`//`) at the beginning of the line instructs the browser to ignore that entire line.

```
// This rectangle is the button that starts the game.
rect(20, 100, 50, 100);
```

2. Block quotes
   A forward slash and an asterisk (`/*`) will start the comment block and the reverse, an asterisk and a forward slash (`*/`), will end it. The browser will ignore everything in between, which can be multiple lines of notes.

```
/*
This is a reminder that the code below is not complete yet.
It might be improved by taking this code and making it into
a function of its own.
*/
```

You will see comments used in this exercise to ignore lines of code that are incomplete or contain errors.

p5.js drawing functions

We will address functions in more detail later but here is a brief explanation. A function is multiple lines of code that achieve a specific task. These are grouped together and given a name so that they can be used again and again.

Later on we will write our own functions but, for now, we will use some functions that are provided by the p5.js library.

- `createCanvas(800, 450)`
  This is called inside `setup()` to create a drawing area of a certain width and height – in this example the canvas is 800 pixels wide and 400 pixels high.

> Within the p5.js library a [HTML canvas element](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API) is created.

- `rect( 50, 100, 200, 40 )`
  This function draws a rectangle 50 pixels from the left of the canvas (`x`), 100 pixels from the top (`y`). The width of the rectangle will be 200 pixels and the height will be 40 pixels.

Cartesian Coordinates

To understand how to position elements on screen we need to go back to school. When drawing to a screen on the majority of programming languages will use a version of the Cartesian Coordinate system.

It was a system developed in the 17th Century by René Descartes for locating unique points on a mathematical representation of a 2D plane using numerical pairs; e.g. `(50, 100)`, `(251, 122)`. This revolutionised the fields of geometry and algebra centuries before the first computer screens.

For our purposes, the numerical pairs represent the number of **pixels** counting from left to right (`x`) and top to bottom (`y`). For most, the diagram on the left will be familiar for plotting points on a graph:

drawing-03.png
(image credit: https://processing.org/tutorials/pixels/)

The only difference between plotting points on a graph and on a screen using code is that (in nearly all languages) we plot points on a screen starting from the top left corner rather than the centre. You need an `x` value (horizontal position) and a `y` value (vertical position) in order to specify a pixel position on screen.

Using coordinates in functions

In our code we call the following function:

```
rect( 50, 100, 200, 40 );
```

The function accepts 4 arguments that define the position and shape of the rectangle:

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

Therefore the result of this will be the following:

cartesian.png

Each function in the library can take different arguments depending on its purpose. For example, when defining a line we do not specify the width and height because lines are 1 dimensional (they have zero or negligible height). Instead, a line is better defined by a *start* and *end* position on our screen; two sets of Cartesian Coordinates:

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

Below is a diagram showing how this using the cartesian coordinates system.

drawing-06.png(image credit: https://processing.org/tutorials/pixels/)

You will not be expected to instinctively know what arguments to give to a particular function like `line()` or `rect()`. When using libraries written by someone else, it is common for the authors to provide online documentation describing each of the functions.

Documentation

We know from our sketch that the `rect()` function accepts a minimum of 4 arguments: x, y, width and height. Without being told, how do we know what these parameters mean? And what about other functions like `triangle()` or `quad()`?

To find out, we check the online documentation provided by the authors of the library or programming language. You can search online for the function you are using and the documentation will give you all the information you need to use it, typically with some useful examples. We can check the **reference for p5.js**, and specifically the page that explains the **line function**.

Exercise

- Add the `01_coordinates_and_shapes` directory to Atom
- Open `index.html` in a browser
- Change the position, width and height of the rectangle
- Draw a line
- Draw an ellipse, triangle, or quad

Colour

The code to support this section is located in the following directory and is available to view on **Github**:

```
/02_colour_stroke_fill/
```

RGB Colour Space

When defining a colour in code, we need to describe it numerically using a 'colour space'. The most common colour space used in computing is RGB (Red, Green, Blue). Those with experience of graphical software such as Photoshop will be familiar with the colour selector that shows you the RGB values as your move around the colour palette:

selector.jpg

An RGB colour can be understood by thinking of it as all possible colours in the visible spectrum that can be made from combinations of red, green, and blue light. By defining the intensity of each of the three colours that are mixed together, it's possible to pick from over 16 million different colours. Arguably **more than the human eye can see**.

In practical terms, we specify the individual amounts of red, green, and blue using values between **0 and 255**.

For example, this describes the colour red:

```
255, 0, 0    <---- RED
```

This describes green:

```
0, 255, 0    <---- GREEN
```

And this describes the orange colour used on this website:

```
255, 152, 0    <---- ORANGE
```

Additive colour

rgb.jpg

In contrast to **subtractive** colour models, such as CMYK used for paints and print, the RGB colour space is **additive**. When you mix the primary paints or pigments together the resulting colour will become increasingly dark, working its way towards black. With colour displayed on a computer monitor or mobile device, adding red, green and blue together will provide white.

> If you want to know all there is to know about colour theory then read Joseph Alber's amazing book, **Interaction of Colors**.

Using colour functions

In the p5.js library there are functions provided for controlling the colour of the fill and stroke of shapes.

- `fill(r, g, b)`
  This determines the main body of colour inside a shape.
- `stroke(r, g, b)`
  This defines the colour of the line that surrounds the shape.

Here are some examples of giving three arguments (r,g and b) to the fill and stroke functions:

1. `fill( 255, 0, 0 )` // red shape fill
2. `fill( 255, 255, 0 )` // yellow shape fill
3. `stroke( 0, 0, 255 )` // blue outline
4. `stroke( 255, 0, 255 )` // magenta outline

Grayscale

Another feature of these functions is the ability to use them to define grayscale values. Passing a single argument between 0 and 255 will result in a colour between black and white:

1. `fill(0)` // black shape fill
2. `fill(255)` // white shape fill
3. `stroke(150)` // grey outline

Order is important

When calling these functions you are defining the fill and stroke colour for all the shapes you draw after that line of code. So it is important to pay attention to the order in which you use them.

Exercise

The code below draws a selection of shapes around the canvas. They are all coloured white, gray or black. Your task is to add some colour to this situation.

- Add the `02_colour_stroke_fill` directory to Atom
- Open `index.html` in a browser
- Change the fill and stroke colour for each shape

Simple Interaction and variables

The code to support this section is located in the following directory and is available to view on **Github**:

```
/03_simple_interaction/
```

p5.js defines some variables that we can use in our code about the properties of the sketch and also user inputs (e.g. mouse and keyboard). We can use these to make our code easier to maintain, more flexible, and to possibly add some basic interactions.

What is a variable?

A variable is how we store useful values in code. The *types* of things we can store depends on the programming language being used, but common examples are numbers and text.

Think of a variable as a container or box. The value is the thing inside the box, and the label on the front of the box is the name we use to identify it.

```
var myNumber = 5;
var myText = "hello";
```

In reality, the variable's container is a small section of memory on your computer.

p5.js provided variables

After you've called the `createCanvas(width, height)` function, p5.js automatically stores the specified dimensions as variables named `width` and `height` that can be used throughout your sketch. For example, you can use those variables to calculate and draw something in the exact centre of the canvas:

```
rect(width/2, height/2, 20, 20);
```

Special variables, such as `mouseX` and `mouseY` are made available by p5.js. These are extremely useful if we want to make sketches that respond to the user's mouse input. These two variables contain the `x` and `y` coordinates of the user's mouse at that precise moment. We can use changing values to modify our drawing and create something more dynamic.

Exercise

- Add the `03_simple_interaction` directory to Atom
- Open `index.html` in a browser
- Change the provided code so that a shape follows the mouse around the canvas

Week 1 Assignment

For the next workshop, I would like you to make a portrait (self or other) using what you've learned from week 1. You should use the following functions and variables:

- `rect()`
- `ellipse()`
- `triangle()`

- `fill()`
- `stroke()`
- `mouseX / mouseY`

I would like you to use **Codepen** to submit your work. Codepen is an online code editor for web based technologies (HTML, CSS & JavaScript) as well as a platform for sharing your code. I have created a template for you to use that already includes the p5.js libraries:

http://codepen.io/pen?template=zKLpKw

Follow the link above and then edit the code in the JS panel. Click **Save** and you will have created a 'pen' with a unique URL (see below). **Submit the Codepen URL to our Slack channel before the next workshop.**

Codepen - Create Pen from template

Revision #41
Created 12 October 2016 15:08:43
Updated 20 August 2018 11:13:01