Living Data Workshop #3 Task (weeks 9/10)

What to do before the workshop

1. Work through the activities in this sheet
2. Complete the assessment on Canvas titled “Living Data Pre-Workshop Task #3” before coming to class.

In the prework

You will learn about showing your data with colour, and making more complex drawings with data.

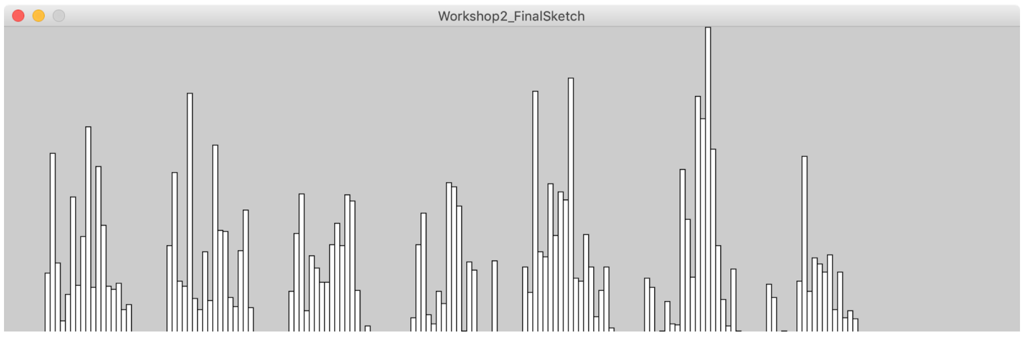
In this workshop

We will do a worked example of the pre-work as a recap.

We will introduce **conditional statements** that allow you to make decisions about which pieces of code will run.

Workshop 3 Starter Pack

During the Workshop last time, we finished the exercise with a bar chart, drawn with code



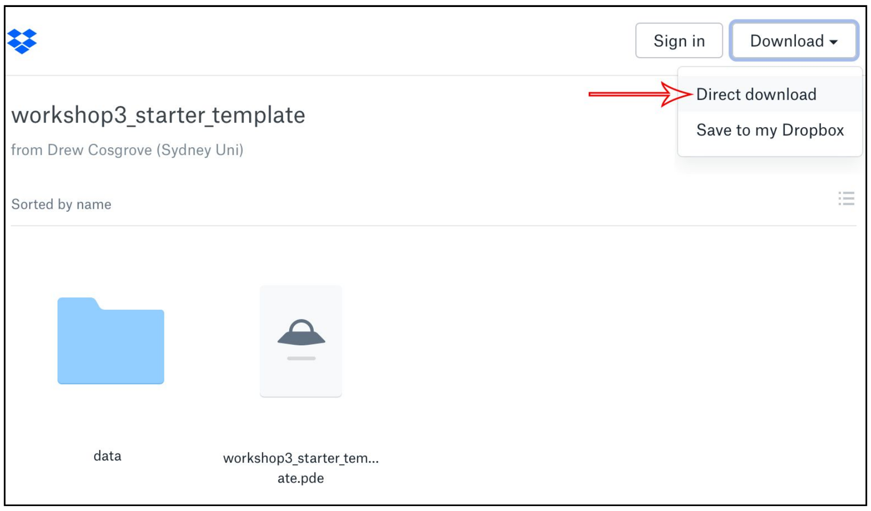
To do this, we combined the ideas of loading data from the Workshop 2 pre-work exercise, with the idea of drawing with code from Workshop 1. If you didn't complete this exercise, including flipping the bar-chart upside down, a completed example sketch can be downloaded from http://bit.ly/LDWS2-FINALCODE-19

This week, we will be drawing with a different set of data - this includes daily step totals for a 7 week period. We will also create a different type of visualisation called a **heatmap**.

To do this, we'll build on the final example in last week's exercise and introduce a few new concepts.

To get started on this workshop, we've provided a Processing sketch file, preloaded with 7 weeks of sample data. Download this from http://bit.ly/LDWS3-CODE-19

Remember in Dropbox to use the **"Direct Download"** option, otherwise you will not get the sketch and the data files required to complete this exercise.


Sign in
Download
Direct download
Save to my Dropbox
workshop3_starter_template
from Drew Cosgrove (Sydney Uni)
Sorted by name
data
workshop3_starter_tem...ate.pde

The `color` data type

Back in Workshop 1 we introduced the idea of RGB Colour. Processing has a data type specifically for working with colours, and this enables us to hand of some of the complexity of dealing with colour. This new data type is called **`color`**[1].

The `color` data type takes in three values. One each for red, green and blue. You need to use the helper function `color()` in order to use it.

```
        color myColor = color(red, green, blue);
```

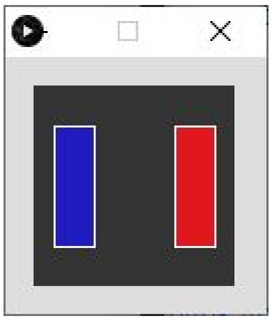
Let's explore how this works. Open the **workshop3_color_example** sketch file from this week's download.

This example includes two lines of code that are unfamiliar. These define two `color` variables called `from` and `to`:

```
        color from = color(32, 27, 191);
        color to = color(224, 23, 29);
```

`from` is a shade of blue (note the high **blue** parameter). `to` is a shade of red (note the high **red** value).

When we run this example, you get this result:



Activity: Try changing the **R, G and B** parameters for the two colours. Make sure you know how these affect the colours. Remember that you can use the Color Selector in Processing (in the drop-down menu > Tools > Color Selector) to help choose the red, green and blue numbers that functions like `color()`, `fill()`, `stroke()` and `background()` need to work properly.

---

[1] Processing (and pretty much every programming language) was developed by a group of American programmers. Therefore it uses the American spelling of "colour".

lerpColor

The `lerpColor()` function (https://processing.org/reference/lerpColor_.html) in Processing lets us do an *interpolation* between any two colours. Interpolation is similar to the idea of mapping we looked at last time. It lets us provide any two colours and it will create new colours that lie somewhere in-between. If you have seen a gradient before, this is the same idea.

In the image below, we start with red on the left, have a orangey-brown in the middle, and end up with green on the other end:

You could also transition between black or white - i.e. no colour, and a colour:

This is useful to us as it gives us the ability to accurately map our data values into range of colours.

While we could use the `map()` function to achieve this, we would have to calculate a map value for each of the red, green and blue colour values. `lerpColor()` will do this for you.

Adapting from the Processing Reference, lerpColor() takes three parameters.

Syntax		`lerpColor(c1, c2, amt)`
Parameters	`c1`	color: interpolate from this color
	`c2`	color: interpolate to this color
	`amt`	float: A number between 0.0 and 1.0

Extending from the code example we looked at before (**workshop3_color_example**) we can add some new code to make this work.

Let's use the `lerpColor()` function, using `from` and `to` as the two colours we want to blend between, and we will provide a number which represents how far along the gradient between the two colours we want Processing to calculate.

At the bottom of the code we'll add these two lines:

```
color interA = lerpColor(from, to, .33);
color interB = lerpColor(from, to, .66);
```

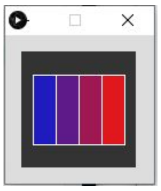
This will calculate two new colours, the first `interA` will be 33% between blue and red, and the second `interB` will be 66% of the way between these colours.

If you then add these two lines, we can draw rectangles to show what those colours look like:

```
    fill(interA);
    rect(30, 20, 20, 60);

    fill(interB);
    rect(50, 20, 20, 60);
```

And running your code will draw four rectangles showing the four colours next to each other.



Activity: Try changing the percentage (the `amt` parameter) and see how this alters the colour that `lerpColor()` will produce.

Combining this with map()

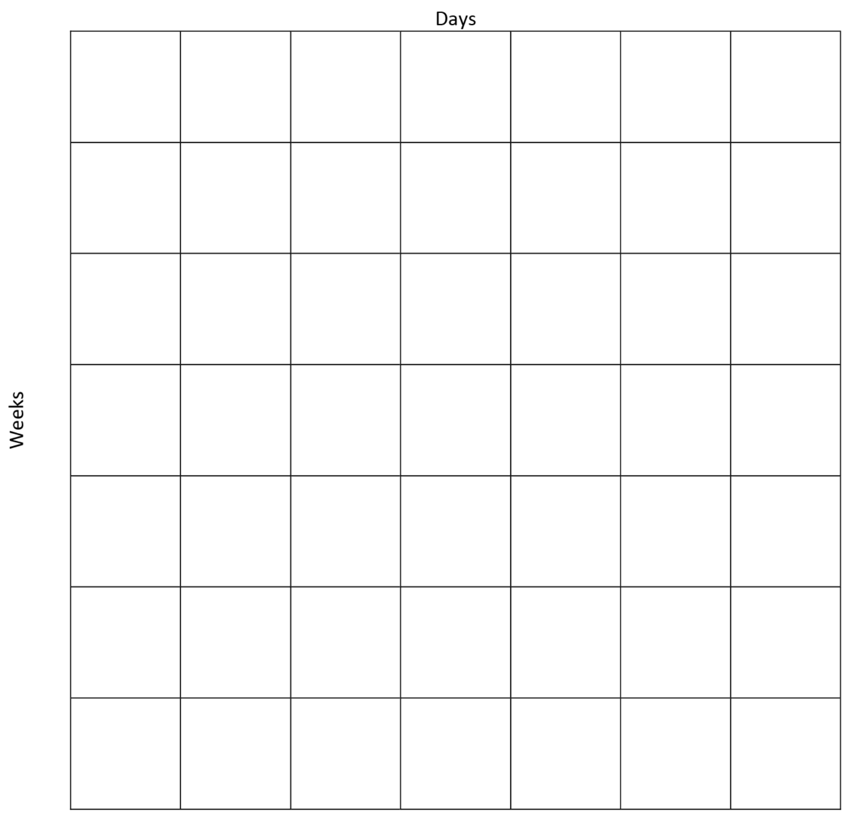
Hopefully by now you have noticed that the `amt` parameter that `lerpColor()` accepts needs to be a number between zero and one. But our data doesn't fit between zero and one. How will we *map* our data range?

The answer of course is the `map()` function we learned in Workshop 2. Think about how to take the minimum and maximum values of our new data and map them to the range between 0 and 1.

Final Activity: Heatmaps

A heatmap is a 2-dimensional matrix of data, with the difference between the smallest and largest values encoded as an intensity of a colour, or as a smooth gradient between colours.

The heatmap we create will display our data with *days* displayed along the x-axis and *weeks* on the y-axis.:



The the number of steps per day will be shown on the heatmap as the colour, or intensity of colour in each box.

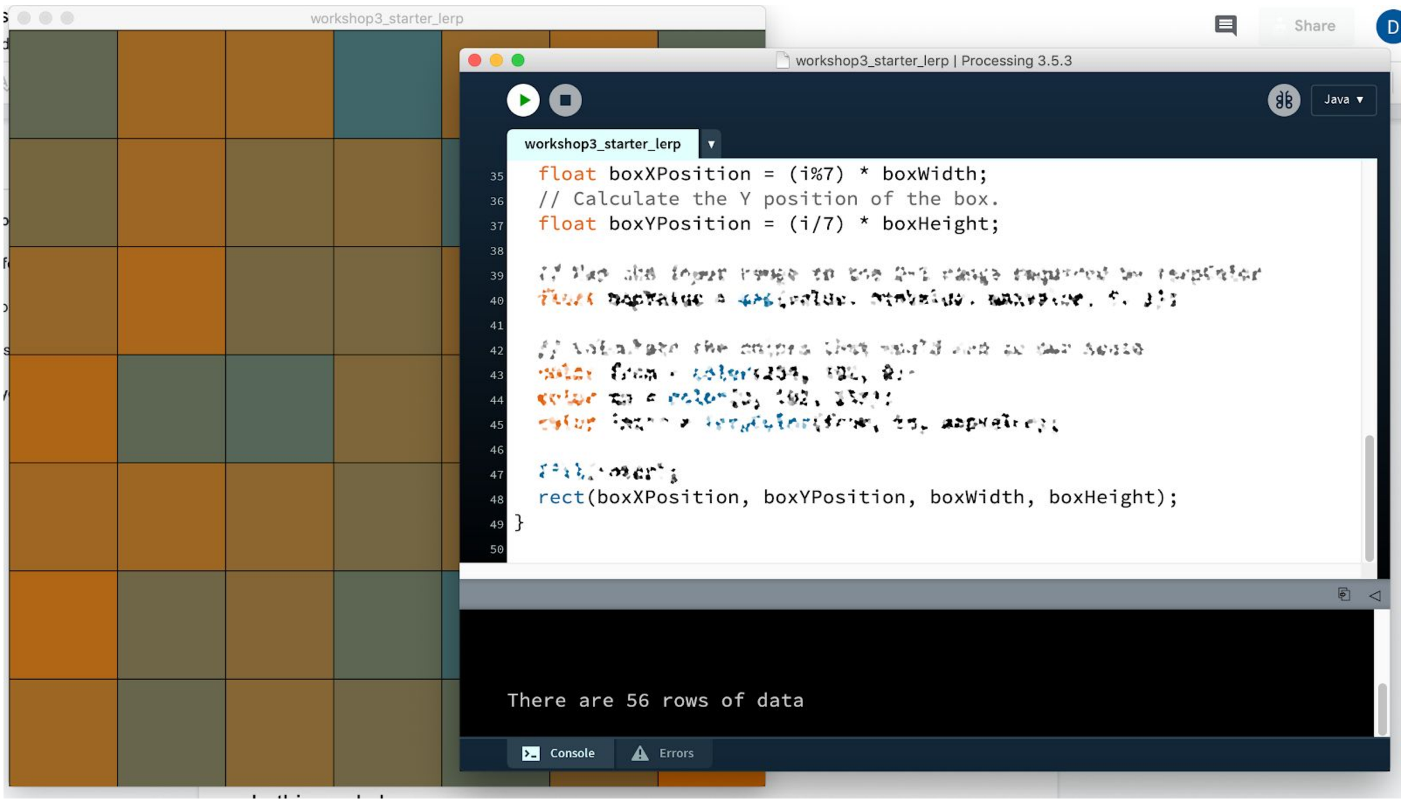
In the starter template that we have given to you, the code that draws the columns and rows is already written.

Your job is to complete the code in **workshop3_starter_template**:

- **Step 1:** Identify the relevant code from the Workshop 2 example (you can refer to the slides to understand what's going on) to **load** the supplied data file (**ios-daily-subset.csv**) and copy this into the starter template
- **Step 2:** Identify the relevant code to determine the **minimum** and **maximum** values in the "**Steps (count)"** column in the data, and copy this into the starter template
- **Step 3:** Use the `color` data type and `lerpColor()` function to map each data point into colours on the heatmap

Once you have this working, take a screenshot (see the example below) showing two things:

- **Your heatmap** (the Processing sketch window); *and*
- **Your sketch showing the `lerpColor()` code**



```
  float boxXPosition = (i%7) * boxWidth;
  // Calculate the Y position of the box.
  float boxYPosition = (i/7) * boxHeight;

  [redacted]
  rect(boxXPosition, boxYPosition, boxWidth, boxHeight);
}
```

```
There are 56 rows of data
```

Marking Criteria

0 marks - No submission or attempt, or a submission showing the same colour codes from this example

1 mark - Draw the heatmap, but without applying colour

2 marks - Implement the lerpColor() code as given in the worksheet

3 marks - Use your own two colours

If you don't get the three marks - bring your code to class and we'll demonstrate how to complete this fully.