THE UNIVERSITY OF SYDNEY

BIOL1008 Workshop 1

Tables and Loops

Visualising data using Processing

What to do before Workshop 1:

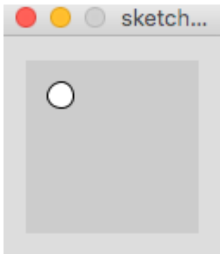
1. Download Processing from https://processing.org/download/ (you do not need to make a donation, if it asks) if you haven't done so already. If you can't run Processing on your computer:
    a. Try to download processing onto a usb drive, and run it from there.
    b. You can run Processing in a web browser at https://www.openprocessing.org/ (registration or sign-in required, but quality interface) or http://sketchpad.cc/ (no sign-in required)
2. Complete this worksheet
3. Complete the pre-work quiz on canvas. It is worth 2% of your total unit mark.

Processing Recap

Let's recap some of the things we saw in the video about processing. Here we have a **function** that draws an ellipse. It's a program in a single line!

```
ellipse(20, 20, 15, 15);
```

If we type this into Processing and press the run button, Processing will open a window with a canvas and draw a small circle near the upper-right corner.



If we change the **parameters** of the function in between the `( )`, and press run again, it will draw in a different place or with a different size. Try it on your computer.

The ellipse function

```
ellipse(x, y, w, h);
```

Draws an ellipse, where:

- x is the position of the center of the circle from left to right (x-axis)
- y is the position from the top of the canvas down (y-axis)
- w is the width of the ellipse
- h is the height of the ellipse

It's difficult to remember every parameter for every function, so remember that you can look up a function's use in the reference at:

https://processing.org/reference/

One other thing to recap here is `setup()` and `draw()`. If you unsure of some of the terms we use, you can find some of them at http://hello.processing.org/guide/

Setup
The block of code after `setup()` is run once (only) at the start of the program. It is useful to put some functions here that make changes to our program that won't be updated at all. This may be:

- Changing the size of the canvas we can draw onto using the `size(x, y)` function
- Setting the `stroke()` or `fill()` colour if we don't need to change it again.
- Loading data that we wish to visualise (more on this later)

Draw
This block of code after `draw()` is run continuously after `setup()` has completed. It's where most of the work gets done, such as animating objects.

A simple program using both of these might look this: (copy it to your own processing window)

```
// hint: everything on a line after //
// is a comment, and won't be run by processing
// it's useful for making notes in your code

float y = 300;      // declare variable, y, with a value of 300
float x = 0;        // declare variable, x, with a value of 0

void setup() {      // code between these { } runs once, first
  size(800, 600);   // set size of the sketch to 800x600 pixels
  fill(200, 0, 255);// set the inside colour to purple
}                   // end of setup()


void draw() {       // code between these { } runs continuously
  background(255);  // draw a white background
  ellipse(x, y, 10, 10);  // draw a circle
}                   // end of draw()
```

We do a few things here that weren't in the video: we make x and y **variables**. They are used store a number for us, but can store many types of thins. In our sketch, whenever we use `y`, it substitutes 300 (or whatever you changed it to). Remember to give your variables meaningful/sensible names, since they are helpful for you. It will make your coding life a lot easier.

Can we make the circle move across the screen?
Try adding this line after your code draws the `ellipse()` before the end of draw block.

```
x = x + 1;
```

This changes the value of x by adding 1 to whatever it happens to be at the moment, so when we draw the circle again, it will be slightly to the right, and it will move across the screen (and keep going forever). Because x is declared at the very start of our sketch, outside of any `{ }` when we change x, that change is there the next time processing redraws the circle. If you change the value x increases by, you can change the speed it moves across the screen. What happens if you change the 1 in this line to 5, or 0.2?

What if we want it to wrap around the screen?

We can use the `if()` statement that we saw in the video to make the circle go back to the start after reaching the edge.

Add this code after you increase the value of x (before the end of `draw()`)

```
        if (x > width) {
          x = 0;
        }
```

Processing will now check to see if `x > width` after adding 1 to the value of x. If this is **true**, then it changes the value of `x` to `0`. If it is **false**, the code in the block after `if()` is not run at all. Keep your sketch open, or save it for the next section.

Loops

According to Larry Wall (in *Programming Perl*, 2nd Edition, O'Reilly & Associates, 1996), the original author of the Perl programming language, one of the three great virtues of a programmer is *Laziness*.

> **Laziness**: The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful and document what you wrote so you don't have to answer so many questions about it.

Drawing one circle is easy. Maybe even drawing 5 or 10. But what if we want to draw 50 circles? 1000? Sure we can write LOTS of code to draw lots of circles, but what's the easier way?

Introducing: **Loops**.

Replace the `ellipse()` function with these lines of code in your processing sketch:

```
      for (int i = 0; i < 5; i++) {
        float yOffset = i * 12;
        ellipse(x, y + yOffset, 10, 10);
      }
```

Now we see 5 circles being drawn. Why? Because the `for` loop tells the program to repeat a block of code.

But first, we see three statements inside the `( )`, separated by semicolons, `;`

- `int i = 0`
  The first statement creates a variable called `i`, with a value of `0`, that only exists inside the `{ }` of the loop. We don't *need* to call it '`i`', but that's the convention we will stick with for now.

- `i < 5`
  The loop will keep repeating as long as the second statement is `true` (that is, as long as `i < 5`). If we change the 5 to 10 or 50, we will see more circles. We could also use a variable here, instead of a number.

- `i++`
  The value of `i` increases by `1` after each loop. If we increase `i` then it will eventually become greater than 5, `i < 5` will become `false`, then the loop ends.
    - If we want to increase by 2 (or any other number) we can change the last statement inside `for()`. Try substituting `i++` for this and see what happens

        - `i += 2`
    - **Tip**: the following statements will have the same effect on the value of `i`:
        - `i++`
        - `i += 1`
        - `i = i + 1`
    - **Challenge**: Our sketch includes the line `x = x + 1;`
      Can you update this line now?

The code **block** inside `{ }` is being repeated, but each time, the value of `i` increases. We use this to increase the value of `yOffset` for each circle we draw. We are using the variable `i` to move the circles away from each other, what happens if we don't add `yOffset` to the y value in our ellipse function?

This is a short introduction to `for()` loops. They're very useful in programming, because they reduce the lines of code that it takes to do repetitive drawing.

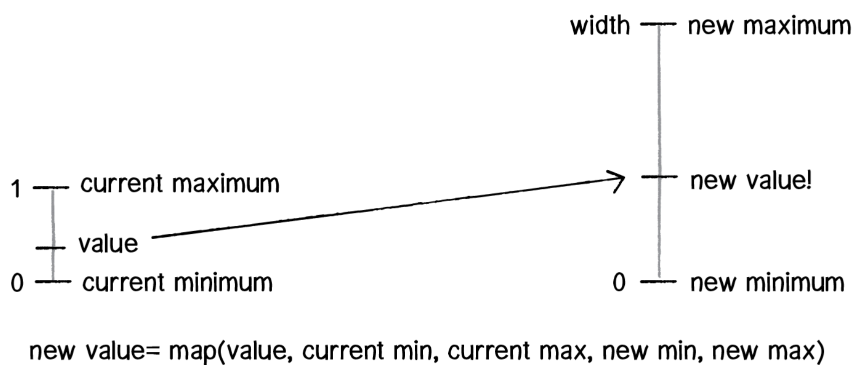
Tables

If we have data in an excel spreadsheet or a .csv file (like a simplified excel spreadsheet), it's very easy to bring data into Processing and use it to draw. But first there is one more function to learn: `map()`.

```
map(x, currentMin, currentMax, newMin, newHigh);
```

Where:
- `x` is the value you want to scale
- `currentMin` is the current minimum for the x-scale
- `currentMax` is the current maximum for the x-scale
- `newMin` is the new minimum for the x-scale
- `newHigh` is the new maximum for the x-scale

In one of his books, Dan Shiffman (our friend from the video) shows us:



(*The Nature of Code* by Daniel Shiffman http://natureofcode.com/book/introduction/)

The map function takes an input, and scales it from one range to another:

```
map(0.5, 0, 1, 0, 100)
```

returns 50, and

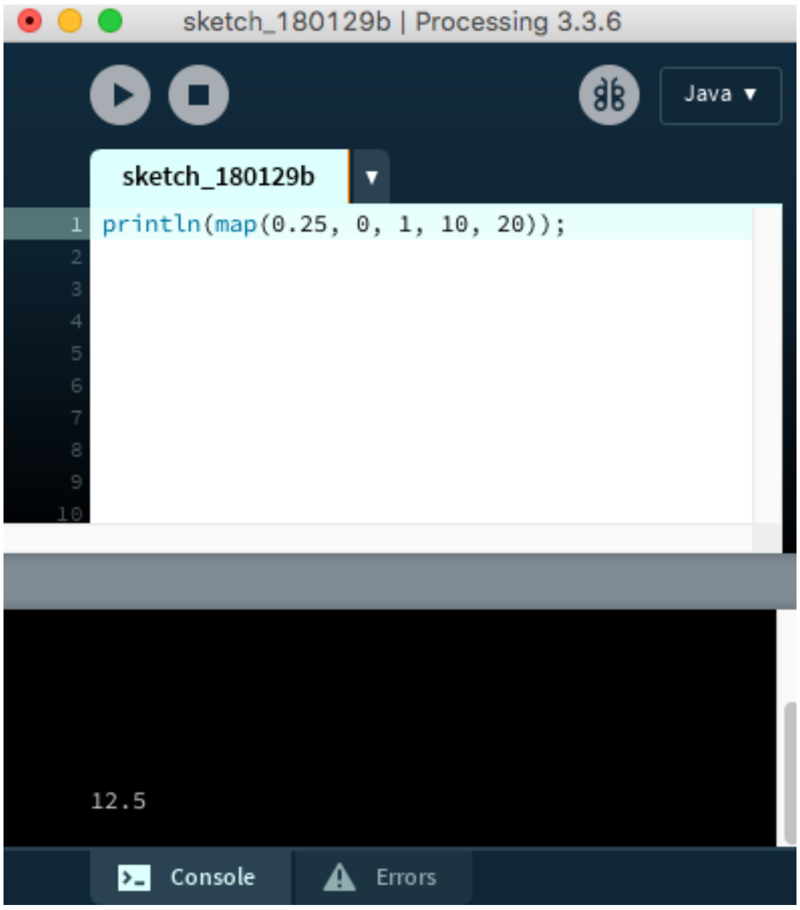
```
map(1.25, 1, 2, 10, 20)
```

returns 12.5.

Try it for yourself, open a new Processing sketch, copy this function and play with the values of the numbers to get the hang of it:

```
        println(map(0.25, 0, 1, 10, 20));
```

Tip: The `println()` function will evaluate and print whatever is in the parentheses into the console (the black area) at the bottom of the Processing window.



```
println(map(0.25, 0, 1, 10, 20));
```

```
12.5
```

In the folder for this workshop, there is a folder called processing1. Inside this folder there is a processing sketch file processing1.pde and a data folder. Inside the data folder there is a data.csv file. This file has some “synthesized” daily step counts.

Have a look at the code from the processing file. **We will be going through it in class.**

- Run it on your computer to see what happens
- See what’s familiar to you and what’s new
- Can you figure out what one of the new functions do?
- When you run the processing sketch, what do you see?
- Can you change the colour of the line we draw?

```
// somewhere to store the data from our csv file
Table table;

// declare a variable to store the total number of days
float numDays;
// declare the maximum steps from our data, we will start at 0
float maxSteps = 0.0;

// setup the sketch, run setup() once at the start
void setup() {
  // tell the sketch window how big it will be
  size(800, 450);
  // add the data to our table. Note: our data does have a header row
  table = loadTable("data.csv", "header");
  // user feedback : how many rows are in the table
  println(table.getRowCount() + " total rows in table");
  // assign the number of rows to our numDays variable
  numDays = table.getRowCount();
  // for each table row...
  for (int i = 0; i < numDays; i++) {
    // temporarily store which day and how many steps you did
    int day = table.getRow(i).getInt("day");
    int steps = table.getRow(i).getInt("steps");
    // was today was the day you did the most steps so far?
    // save the bigger value out of maxSteps and today's steps in maxSteps
    maxSteps = max(maxSteps, steps);
    // user feedback: how many steps did I do today
    println("On day", day, "your step count was", steps);
  }
}
// end setup()

// draw() is the part of the sketch that runs continuously
void draw() {
  background(255);
  // draw a line chart
  // start at int i = 1, since we draw between 2 points
  for (int i = 1; i < numDays; i++) {
    // calculate position of first x, y coordinates
    float x1 = map(i-1, 0, numDays, 10, width - 10);
    // height - map() or otherwise the number will be upside down
    float y1 = height - map(table.getRow(i-1).getFloat("steps"), 0, maxSteps,
10, height-10);
    // calculate position of second x, y coordinates
    float x2 = map(i, 0, numDays, 10, width - 10);
    float y2 = height - map(table.getRow(i).getFloat(1), 0, maxSteps, 10,
height-10);
    // set stroke colour
    stroke(0);
    // draw line
    line(x1, y1, x2, y2);
  }

  // draw circles on top of the line
  // start at int i = 0 to draw a circle at every point
  for (int i = 0; i < numDays; i++) {
    // calculate x, y, position for center of circle
    float x = map(i, 0, numDays, 10, width - 10);
    float y = height - map(table.getRow(i).getFloat("steps"), 0, maxSteps, 10,
height-10);
    // set stroke and fill colour of circles
    stroke(0);
    fill(0);
    // draw ellipse
    ellipse(x, y, 10, 10);
  }
}
// end of draw();
```