

Advanced drawing in Processing

BIOL1008

Workshop 2

Advanced Drawing  
Video Peer Review

# What to do before Workshop 2:

1. Watch the video: <https://youtu.be/C29wJHNUwYA>
2. Work through this sheet
3. Check the folder on Canvas for additional information you will need:
   1. Copy of the workshop 1 slideshow as a pdf
   2. Copy of Phil’s activity and nutrition data
   3. Marking rubric for the video component for the workshops
   4. The pdf of the living data brochure
   5. Updated version of the Processing template used in Living Data workshop 1
4. Complete the pre-work quiz on Canvas

## Using beginShape()and endShape()

When we draw by hand we put the pencil down somewhere, move it somewhere else, and so on. When we finish drawing, we lift up the pencil. We have a way of doing this in Processing (try it on your computer):

beginShape();  
  
vertex(10, 10);  
vertex(20, 10);  
vertex(40, 40);  
vertex(10, 50);  
  
endShape(CLOSE);

The beginShape() function tells Processing that we are about to put the pen down.

The vertex() functions tells Processing where to put the pen down, and repeating this function tells Processing where to move the pen next.

The endShape() function tells Procesing to lift up the pen. If we include CLOSE (make sure you use capital letters!), it will draw a line between the last and the first vertex.

This is useful because we can use a single loop to draw a continuous line or shape:

Table myTable = loadTable("data.csv", "header");

int numRows = myTable.getRowCount();

beginShape();

for (int i = 0; i < numRows; i++) {

int steps = myTable.getRow(i).getInt("steps");

float xCoord = map(i, 0, numRows, 0, width);

float yCoord = map(steps, 0, 1500, height, 0);

ellipse(xCoord, yCoord, 20, 20);

vertex(xCoord, yCoord);

}

endShape();

Hopefully you find this pattern pretty familiar by now! See the video for an application of this, and how we updated our simple line chart from workshop 1.

## Drawing with sin and cos

You won’t need to create animations in this class but drawing over time helps us think about how sin and cos —maths you probably did in year 10 and maybe never thought about again—can be used to create motion.

First things first: **radians**

When we learned SOHCAHTOA in school trigonometry, we were finding the sin/cos/tan of an angle, with the angle going from a scale from 0º to 360º around a circle. But in Processing, sin() and cos() takes a value in radians—a scale from 0 to 2π around a circle. Note: in Processing, 2π is called using TWO\_PI – try it:

println(TWO\_PI);

Radians is a measurement of an angle of rotation, rather than an angle between two lines (like in a triangle). We are using this in our example to rotate a line chart around a circle.

Have a look at the example here for more information and some example code that explains this in more detail:

<https://processing.org/tutorials/trig/>

Drawing with frameCount allows us to see changes over time, like we saw in the video. But we want to see changes over a data set. Let’s look how we drew a circle using frameCount:

void setup() {

size(220, 220);

background(255);

}

void draw() {  
 // the centre of the circle is the middle of the canvas

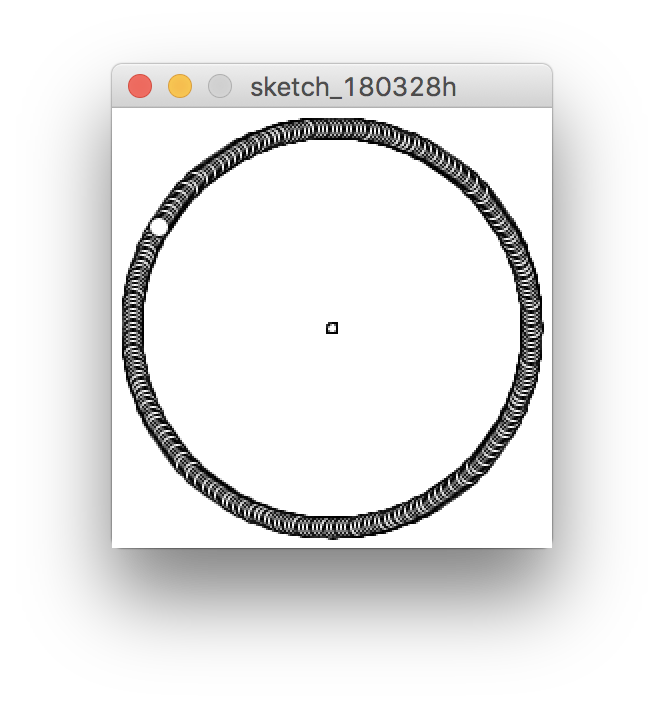
float xPosition = width/2;

float yPosition = height/2;

// draw a small circle at the centre

ellipse(xPosition, yPosition, 5, 5);

// move ellipse around the outside—100 pixels from the middle  
 xPosition += 100 \* sin(radians(frameCount));

 yPosition += 100 \* cos(radians(frameCount));

ellipse(xPosition, yPosition, 10, 10);

}

This code draws an ellipse that moves around a circle with a radius of 100 pixels (try it yourself). It is using the current frameCount as an angle. Since we want to move around every angle on the circle, it works well.

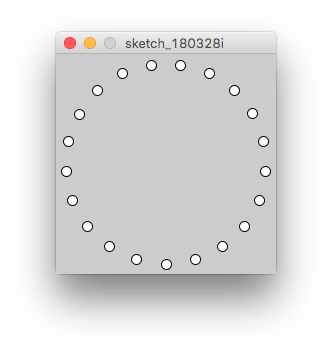
But we may want to draw a few points, based on the number of rows in a data table:

size(220, 220);

// load the sample data from workshop1  
// drag the data.csv file onto the sketch window

Table myTable = loadTable("data.csv", "header");

float numRows = myTable.getRowCount();

for (int i = 0; i < numRows; i++) {  
 // the centre of our circle is the middle of the canvas

float xCoord = width/2;

float yCoord = height/2;

// each ellipse is spaced evenly around the circle

float angle = map(i, 0, numRows, 0, TWO\_PI);

// 100 pixel radius

xCoord += 100 \* sin(angle);

yCoord += 100 \* cos(angle);

ellipse(xCoord, yCoord, 10, 10);

}

We can see in this example that we are able to use map() instead of radians, think about why this is.

The function radians() takes a value, changing its scale from 0 to 360 to 0 to TWO\_PI.

So we could think of radians()is the same as:

float radiansValue = map(someAngle, 0, 360, 0, TWO\_PI);

Just remember:

1. you MUST convert a number to radians before using it on either sin() or cos() by using map() or radians().
2. This will still work correctly if someAngle in the code above is greater than 360.

We can add data to the circle we have drawn in different ways: by varying the angle or the radius of the location where we draw our ellipses.

One way is to use data to make the dots appear closer to the center of the circle when the number of steps is lower. I have highlighted the lines that do this in the code below:

size(220, 220);

// load the sample data from workshop1

// drag the data.csv file onto the sketch window

Table myTable = loadTable("data.csv", "header");

float numRows = myTable.getRowCount();

float maxSteps = 0.0;

for (int i = 0; i < numRows; i++) {

float steps = myTable.getRow(i).getFloat("steps");

maxSteps = max(steps, maxSteps);

}

beginShape();

for (int i = 0; i < numRows; i++) {  
 // the centre of our circle is the middle of the canvas

float xCoord = width/2;

float yCoord = height/2;

float angle = map(i, 0, numRows, 0, TWO\_PI);

// vary the radius based on the data

float todaysSteps = myTable.getRow(i).getFloat("steps");

float dataRadius = map(todaysSteps, 0, maxSteps, 0, 100);

xCoord += dataRadius \* sin(angle);

yCoord += dataRadius \* cos(angle);

vertex(xCoord, yCoord);

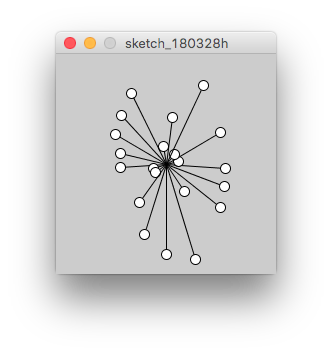
ellipse(xCoord, yCoord, 10, 10);

}

noFill();

endShape();

Instead of our circle being 100 pixels in radius, we change the radius based on data. It shows an absolute picture of how many steps were taken per day.



We could also vary the angle around the circle based on the data:

size(220, 220);

// load the sample data from workshop1

// drag the data.csv file onto the sketch window

Table myTable = loadTable("data.csv", "header");

float numRows = myTable.getRowCount();

float maxSteps = 0.0;

for (int i = 0; i < numRows; i++) {

float steps = myTable.getRow(i).getFloat("steps");

maxSteps = max(steps, maxSteps);

}

beginShape();

for (int i = 0; i < numRows; i++) {

// the centre of our circle is the middle of the canvas  
 float xCoord = width/2;

float yCoord = height/2;

float angle = map(i, 0, numRows, 0, TWO\_PI);

float todaysSteps = myTable.getRow(i).getFloat("steps");

// vary the angle around the circle based on data

float dataAngle = map(todaysSteps, 0, maxSteps, -10, 10);

xCoord += 100 \* sin(angle + radians(dataAngle));

yCoord += 100 \* cos(angle + radians(dataAngle));

vertex(xCoord, yCoord);

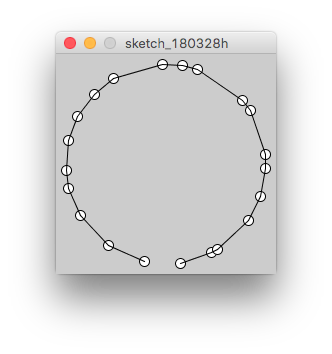
ellipse(xCoord, yCoord, 10, 10);

}

noFill();

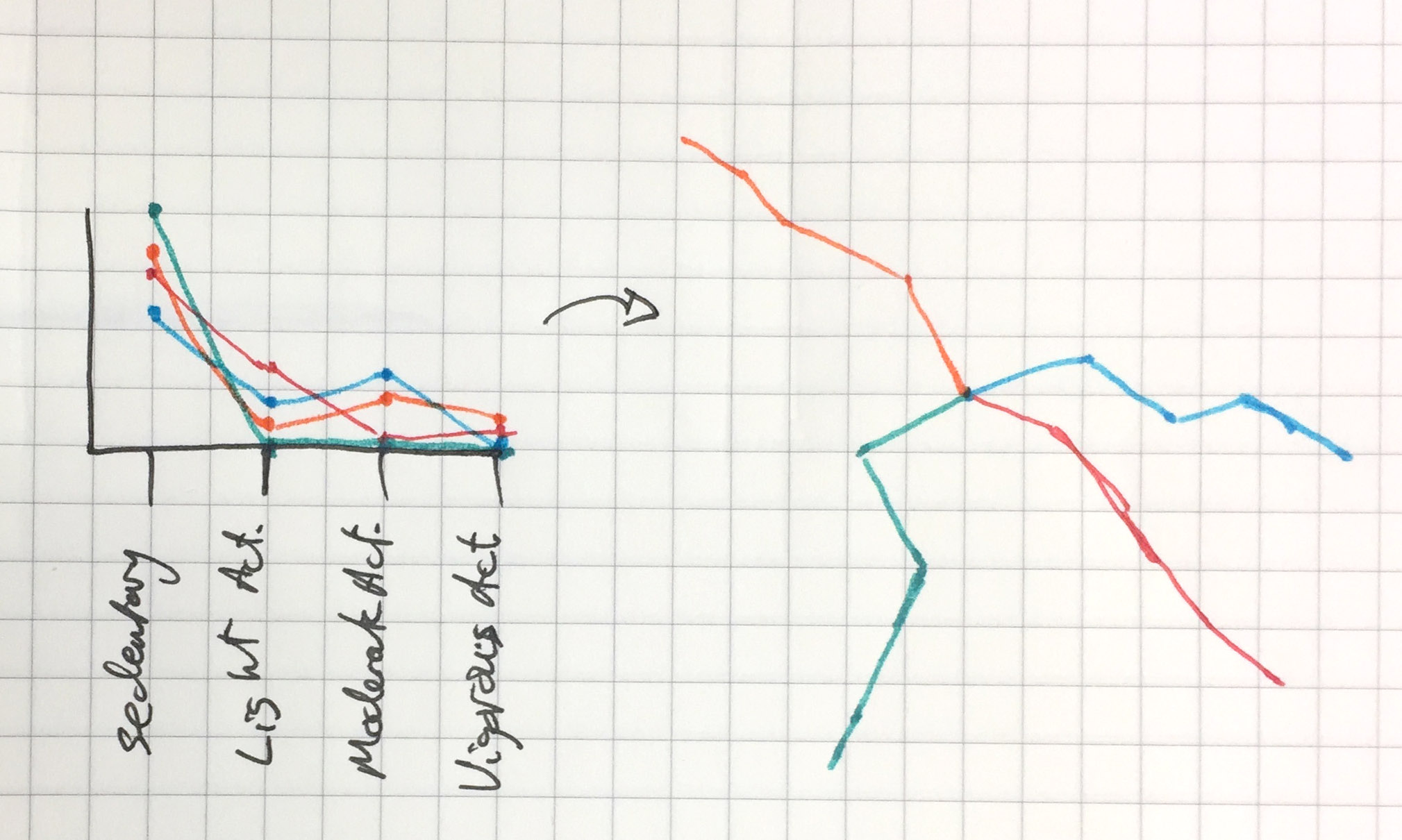
endShape();

This example moves the ellipse around the circle based on data, where 0 steps would move it 10 degrees to the left, and maxSteps (the maximum number of steps in a day) moves it 10 degrees to the right. It shows how many steps were taken *compared to the previous day*.



## Using your (or Phil’s) data in Processing

In the example we transformed a line chart by rotating the lines around a central point, rather than drawing them on the same axis.



We also show only the fill colour, rather than showing the line.

Let’s break down the code. The complete .pde file and the data are both in the folder for this workshop. If you would like to use your own data,

We start by creating a few variables that we will need, loading the data and running our setup:

Table myActivity;

int numRows;

int numDays;

void setup() {

size(800, 450);

myActivity = loadTable("phil\_hours.csv", "header");

numRows = myActivity.getRowCount();

// the total number of days is the number stored in   
 // the last row (numRows-1) in the "Day" column

numDays = myActivity.getRow(numRows-1).getInt("Day");

noStroke();

fill(207, 140,216);

}

The variable numDays takes the data from the last row, in the *Day* column and saves it. We need to access the last row using numRows-1.

Now we begin the draw function by setting a black background

void draw() {

background(0);

Start the for loop to draw our shapes. The first thing we do in this loop is get the data for the current row for the proportion of sedentary activity, light activity and moderate activity and saves them to variables: sedentariness, lightActivity and moderateActivty. (I’ve started new lines because it doesn’t fit in a Word document very neatly.)

for (int i = 0; i < numRows; i++) {

float sedentariness =   
 myActivity.getRow(i)  
 .getFloat("Sedentary Activity (proportion)");

float lightActivity =   
 myActivity.getRow(i)  
 .getFloat("Light Activity (proportion)");

float moderateActivity =   
 myActivity.getRow(i)  
 .getFloat("Moderate Activity (proportion)");

The next part (we are still in the loop) figures out where the centre of the circle will be. We want to know what day it is today, and we can use this to map between 0 and the total number of days (numDays) and a position horizontally along the screen. We will keep them vertically in the middle for now

float dayNum = myActivity.getRow(i).getInt("Day");

float xCentre = map(dayNum, 0, numDays, 100, width - 100);

float yCentre = height/2;

The next is to figure out how far around the circle this line will rotate. We want to map the hour to 24-hour time, so the angle should be mapped from 0 to 24 hours to 0 to TWO\_PI. This will be the direction the line moves from the centre.

float angle = map(i, 0, 24, 0, TWO\_PI);

We next decide two things: how far away the first point will be, and how the data will move the vertex around the circle.

float sedentaryRadius = 25;

float sedentaryAngle = 12 \* sedentaryness;

We use both of these to story an x and y position for a vertex. You will notice that the sedentaryAngle is converted to radians when it is added to the angle.

float sedentaryX = xCentre +   
 sedentaryRadius \* sin(angle + radians(sedentaryAngle));

float sedentaryY = yCentre +   
 sedentaryRadius \* cos(angle + radians(sedentaryAngle));

We then use the same pattern to save x and y positions for a vertex related to light and moderate activity

float lightRadius = 40;

float lightAngle = 15 \* lightActivity;

float lightX = xCentre +   
 lightRadius \* sin(angle + radians(lightAngle));

float lightY = yCentre +   
 lightRadius \* cos(angle + radians(lightAngle));

float moderateRadius = 65;

float moderateAngle = 18 \* moderateActivity;

float moderateX = xCentre +   
 moderateRadius \* sin(angle + radians(moderateAngle));

float moderateY = yCentre +   
 moderateRadius \* cos(angle + radians(moderateAngle));

Then we draw the shape. We could have these throughout the code above, just after these variables are created, but it is clearer to group them all here.

// calculate colours for the shapes - by trial and error

float redVal = map(sedentaryness, 0, 1, 250, 100);

float greenVal = map(lightActivity, 0, 1, 110, 250);

float blueVal = map(moderateActivity, 0, 1, 200, 85);

fill(redVal, greenVal, blueVal);

beginShape();

// start the shape at the centre of the day's circle

vertex(xCentre, yCentre);

// add a vertex for how sedentary I was in this hour

vertex(sedentaryX, sedentaryY);

// and how much light activity I got

vertex(lightX, lightY);

// and how much moderate activity I got

vertex(moderateX, moderateY);

// finish the shape

endShape();

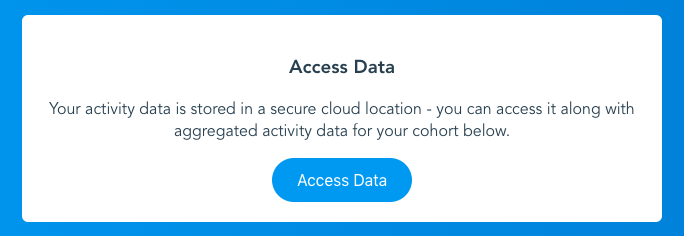
}

}

We will be going through this in more detail during the next workshop.

### Downloading your data

The email you received from *Living* Data has a link to download your data from the activity trackers and intake 24. You can use the *Access Data* link in the email to get to your own data. Note: the data you download will be sample data until a few days after you have worn your activity tracker.



# Video

## Topic, Audience and Style

### Topic: No Stupid Questions

We recommend approaching this problem by looking at a system in the human body.

Remember, there are no stupid questions: only questions for which we haven’t thought about a scientific answer. One good example video we saw in the tutorial asked *Could Humans Ever Breathe Underwater?* —perhaps this may seem like a stupid question, but the video presents an interesting discussion of the related science using a narrative structure.

We don’t want to rule out investigating a ‘stupid’ questions—there are no stupid questions, only questions that we have not yet provided a scientific answer.

### Topic: Start with a System

When choosing a topic, you may want to start by selecting a system. From there, you can ask a question that relates to specific function or pathological state. You may also want to select a topic that you can relate to or have personal experience with.

For example:

Nervous System

* What is the current understanding of the cause of dementia?
* Why does Parkinson’s disease cause a tremor?
* How does the brain control muscle contractions?

Endocrine

* What happens if my dad doesn’t take metaformin with his meals for his type 2 diabetes?
* Why does my mum need to take insulin injections throught the day?

Respiratory

* What does an asthma puffer do?

Gastric

* Why do I get hungry?
* Why would cheeseburgers make you fat?

Circulatory

* What happens when someone has a heart attack?
* Why do people need to take aspirin after getting a heart attack?

Lymphatic/Immune

* What happens after I graze my skin playing soccer?
* What is a cytokine storm? Why is it so bad? When may it happen?

Musculoskeletal

* Why does lifting weights make me huge?
* Why do bones grow when you grow up?
* Why do astronauts come back from space taller?

Perhaps taking a systems approach will help you ask interesting and unusual questions, with interesting topics to investigate scientifically. After you select a topic, you can think about an audience.

### Audience

You are required to select an audience from one of the following:

1. Primary School Students
2. Peers/First Year University students
3. General Public
4. Promoting a government agenda

Check the slides at <https://goo.gl/Q1JHpT> for more detail on these audiences, and links to sample videos. These were discussed in the workshop.

### Style

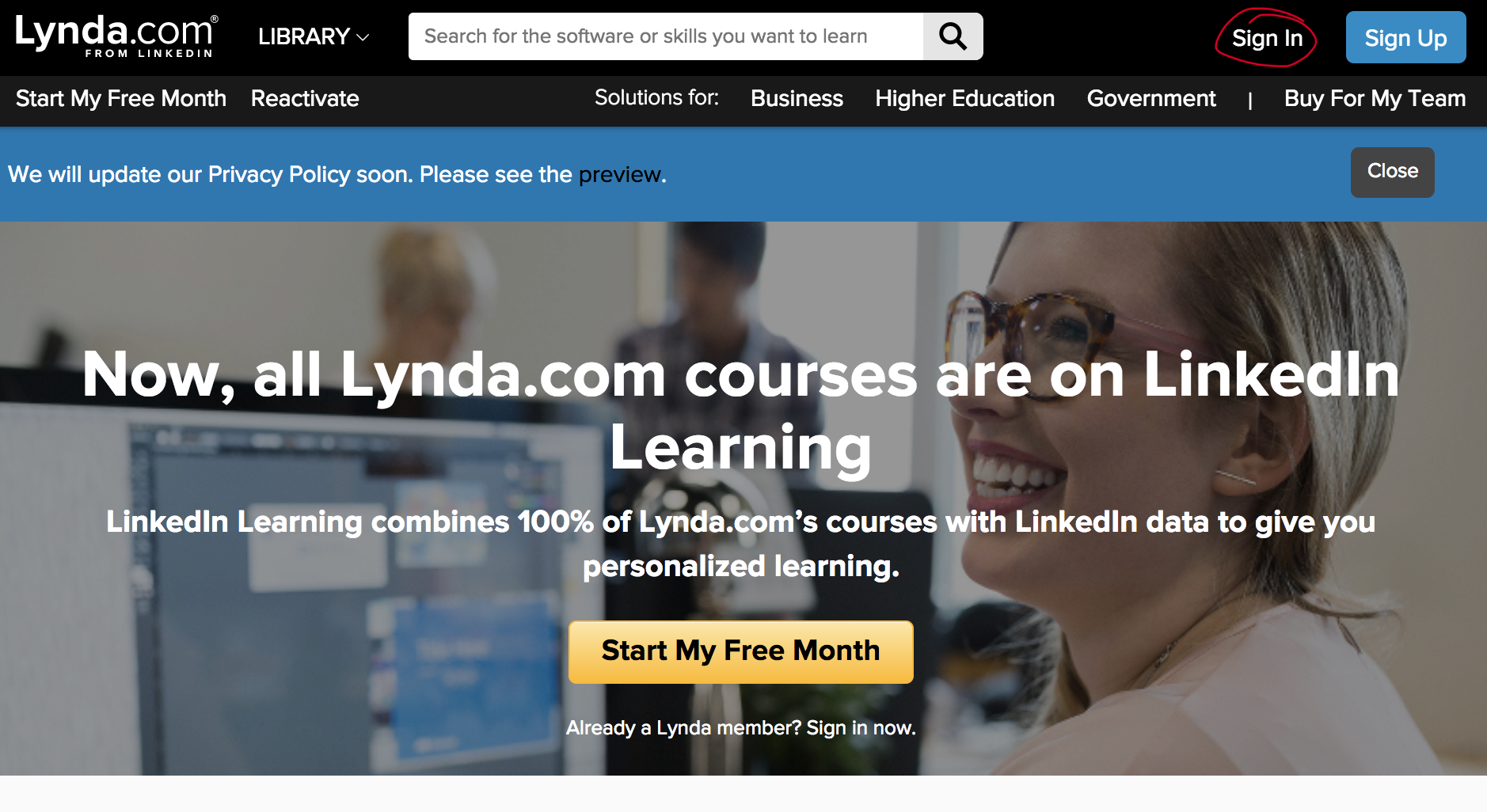
You should still use a narrative structure with any of the video styles. Your video should clearly include **all four elements** of narrative structure: Situation, Complication, Adventure and Resolution.

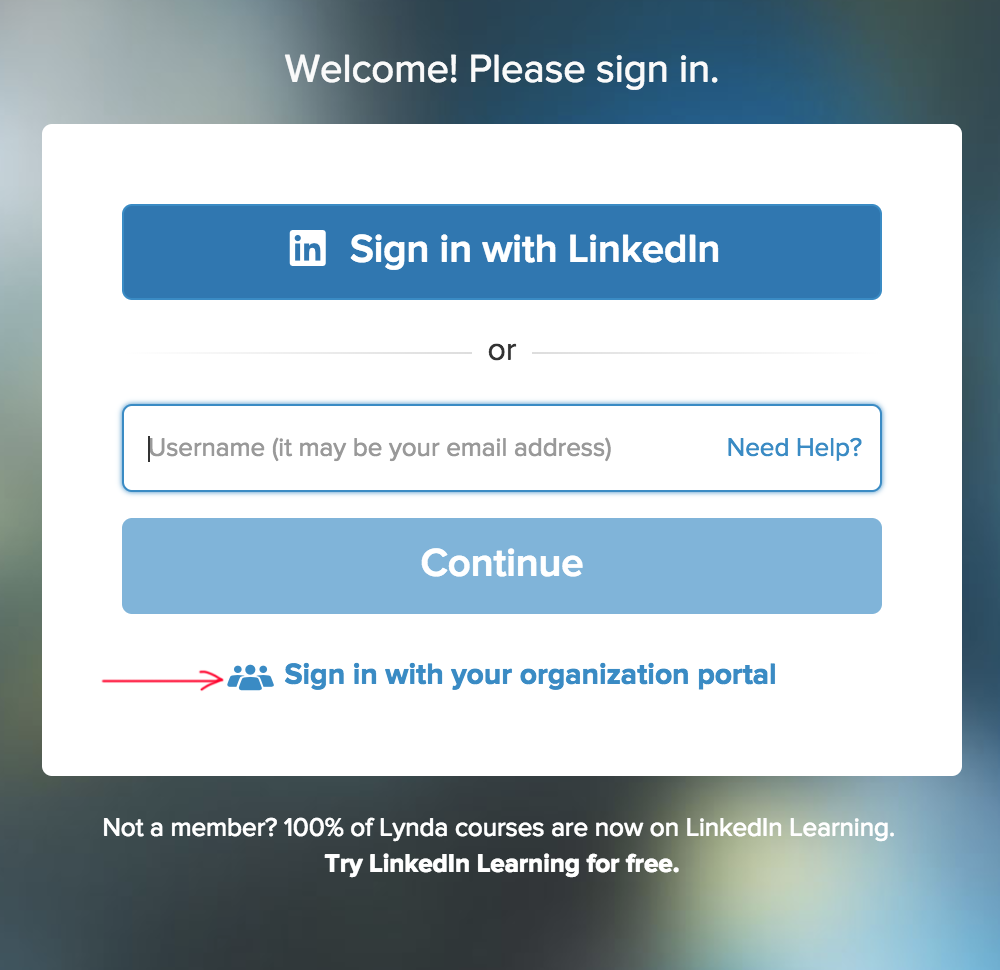
You will select one of the styles that are available in Adobe Spark, <http://spark.adobe.com>. They are also listed in the slides from the first workshop. As we have noted before, you can use another video editing application if you would like, but we will not help you with technical problems that may arise. We strongly suggest that you use Adobe Spark if you have no experience editing video. Whether you make your video in Adobe Spark or not, you must upload your final video as an .mp4 file and also provide a URL to an online version of your video – which is very easy to create in Adobe Spark. If you use your own software, you can upload it to YouTube or Vimeo to provide us with a URL.

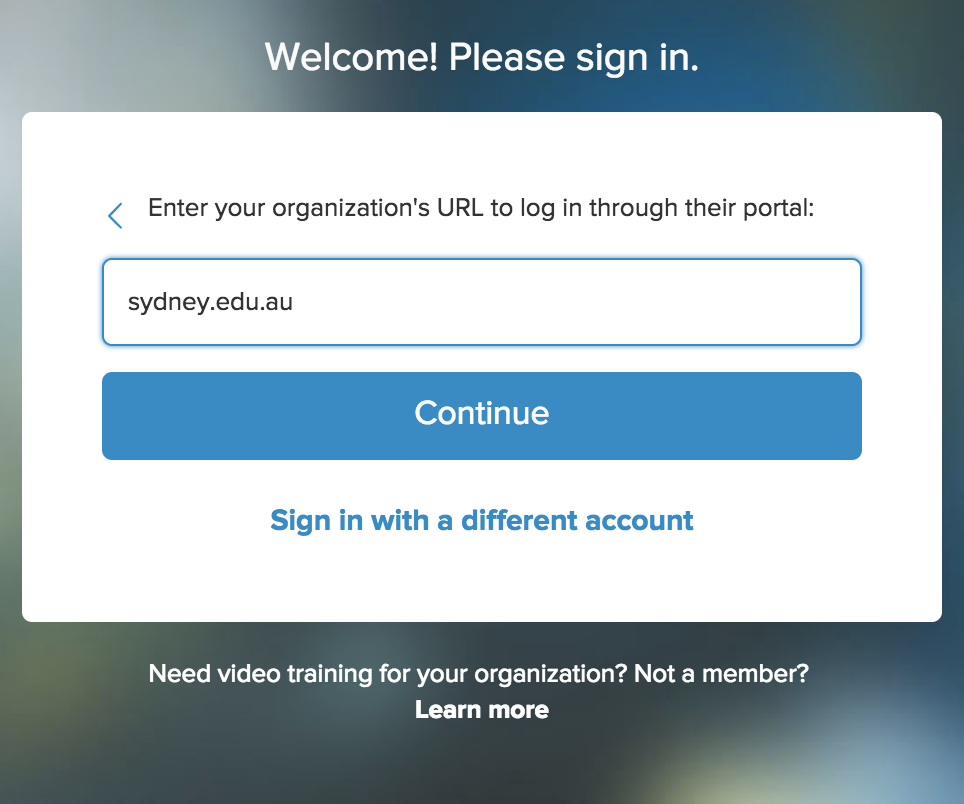
## Lynda.com login

We will be able to help you during class, but if you are having trouble with Adobe Spark, you can access training on Lynda.com. Lynda.com has training for a lot of other software and skills that you may want to learn. To sign in with your Sydney Uni credentials:

Click the *Sign In* link in the top right.

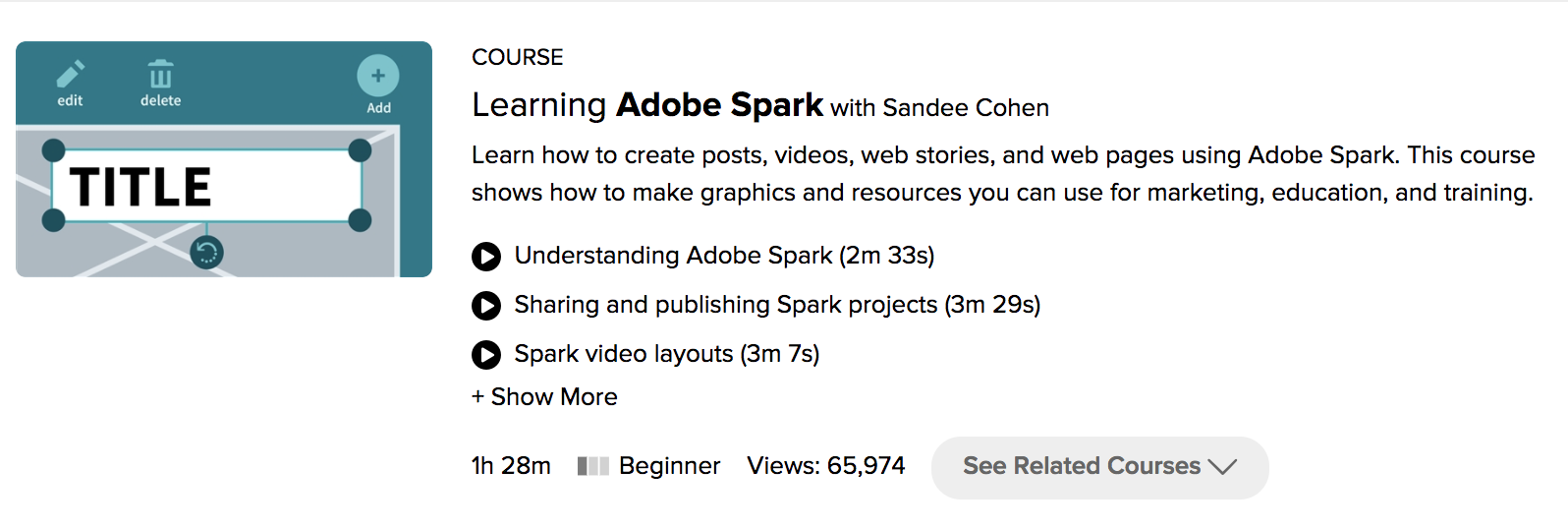


Click the *Sign in with your organization portal* link

Enter *sydney.edu.au* in the box on the next page.

Enter your unikey and password on the Following page. You will be redirected back to Lynda.com

The course you can do is called *Learning Adobe Spark* *with Sandee Cohen*. Chapters 1 and 3 have the information that is relevant to our course.



## Video Plan and Feedback: Marking Rubric

You ***MUST*** bring your draft video plan to the workshop. You will be working in groups of 4 or 5 to share your draft video plan ideas and obtain feedback from each person in your group.   
  
You will be marked on the feedback you give to your peers and you will receive an electronic version of the feedback from your group via email.

Draft video plan (deadline – before second Living Data workshop, submitted through Canvas before midnight on **Sunday night before your workshop**)

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **0 marks** | **0.5 marks** | **1 mark** |
| **Submit your draft video plan**  **(must include chosen topic, audience and style and some ideas on visuals, content and justifications)** | No submission | Draft plan submitted but key lacking sufficient detail – no chosen topic **or** audience **or** style. | Submitted draft video plan, includes topic, audience, style and includes some ideas for the visuals, content, and justification. |

Feedback during second Living Data workshop (deadline – feedback with peers during workshop 2, submitted through Canvas by midnight of the same day)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** | **0 marks** | **1 mark** | **2 marks** | **3 marks** | **4 marks** |
| **Peer to peer feedback on draft video plan in workshop 2.**  **Provide positive, helpful, constructive feedback to peers and gains insight to your video** | No peer to peer feedback provided for any student. | You have provided only **one** type of feedback (positive **or** constructive). | You have provided both a positive element **and** a constructive piece of feedback. | You have provided feedback with both positive **and** constructive elements **and** you have described how you have taken something from it/learned something to apply to your own video plan or video. | You have also provided feedback for everyone in your group (up to 3). |