**Feedback on Session 2 Exercises**

**Task 1**

This should have been a fairly easy task to complete as it simply involved copying code from the lecture notes and making a very simple change to the sketch. You should have replaced the purple text inside the quotes of

|  |
| --- |
| loadImage("sphere.png"); |

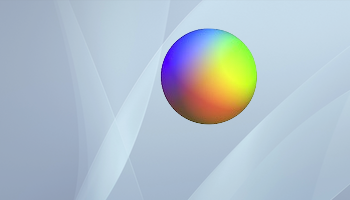
with some URL of your choice pointing to a different image.

Some people tried experimenting with loading a background image and also displaying a smaller image at the current mouse position. To do this you need to make sure that the background image is drawn first, then the smaller mouse-positioned image is drawn afterwards, so it appears 'on top' of the background.

[](http://gicentre.org/datavis/session02/images/sphereOpaque.png)

Overlaid image with opaque background.

Note that once you draw the mouse-positioned image on top of a non-white background, you may have found that if the image you are drawing contains a white background, it too is drawn, producing an ugly result. The solution if you wish to draw a non-rectangular image is to use one with a transparent background. If you do, make sure you save it in a format that can store transparency, such as .png (.jpg files do not store transparency).

[](http://gicentre.org/datavis/session02/images/sphereTransparent.png)

Overlaid image with transparent background

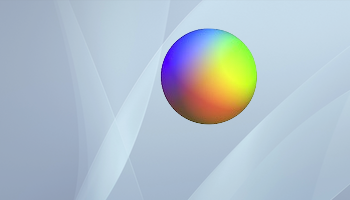
If you had drawn both a background image and a mouse-positioned one you may have noticed that only a quarter of the background image appeared on the screen. This would have been because the command

|  |
| --- |
| imageMode(CENTER); |

which we used to ensure that the sphere appeared centred on the mouse position, also applied to the background image. This means that if you specify (0,0) as the background image position, Processing would actually centre the background image at position (0,0) rather than position its corner at (0,0). A solution is to set the image mode to use corner coordinates before drawing the background image and then set it to use centre positioning before drawing the mouse-positioned image. An example is shown below.

|  |
| --- |
| // Draws a background image and a small image at the current mouse position.  // Jo Wood, 26th January 2016    PImage backgroundImg,movingImg;    void setup()  {    size(400,320);      // Load two images from local files.    backgroundImg =loadImage("background.png");    movingImg     =loadImage("sphereTransparent.png");  }    void draw()  {    // Draw the background image first.    imageMode(CORNER);    image(backgroundImg,0,0,width,height);      // Draw the image centred on the current mouse position.    imageMode(CENTER);    image(movingImg,mouseX,mouseY);  } |

The important lesson here is that settings such as imageMode(), fill() and strokeWeight() will have a permanent effect on your sketch's appearance until you tell it otherwise by issuing one of the commands with a different setting.

[](http://gicentre.org/datavis/session02/imageSketch3/javascript/index.html)

Output from the sketch (click to activate).

**Task 2**

Again, this should have been a fairly simple task as it just involved copying the code from the lecture notes into Processing. To get the sketch to work correctly, you will have needed to drag the two data tables (the country locations and dental health .tsv files) and the background map image into your sketch.

**Task 3**

This task involved modifying the code in Task 2 to use a different GapMinder table of your choice. A few of you had problems loading the file containing the table you had chosen into your sketch. Remember that the file should be saved as tab separated text (e.g. simply saving an Excel file with the extension '.tsv' would not work - you need to 'save as' and select 'tab delimited text'). That file needs to be in the 'data/' folder of your sketch, which you can achieve by dragging it into your Processing code window.

A number of people asked why they couldn't just add the location data directly to the .tsv file containing the data. This would certainly have made the code simple (having only to read a single table), but would require the data table to list exactly the same countries in the same order as the location table. In practice the data in Gapminder had slightly different lists of countries depending on the dataset. Therefore a more flexible approach is to use the country name as a lookup between the two tables as used in the bad teeth example.

While it largely involves simply substituting one table of numbers for another, there were quite a few new programming ideas in the code I provided to do this. I did notice that a few people were getting *Array out of bounds* errors shown when they did this (red text at the bottom of the Processing window). This may be because there was some confusion about which tables to read at what point in the code.

If you were getting confused about what was going wrong, let's just reconsider how we were using tables to plot the GapMinder data for each country around the world. Remember we had two tables, one contained the data we are interested in (e.g. bad teeth) related to each country name. The other related latitude and longitude to country name.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **CountryName** | **NumBadTeeth** | | Afghanistan | 2.90 | | Albania | 3.02 | | Algeria | 2.30 | | Angola | 1.70 | | Anguilla | 2.50 | | Antigua and Barbuda | 0.70 | | Argentina | 3.40 | | *:* | *:* | |  | |  |  |  |  | | --- | --- | --- | --- | | **CountryName** | **CountryCode** | **Latitude** | **Longitude** | | Afghanistan | AF | 33.94 | 65.71 | | Albania | AL | 41.15 | 20.17 | | Algeria | DZ | 28.03 | 1.66 | | Angola | AO | -12.20 | 17.87 | | Anguilla | AI | 18.22 | -63.07 | | Antigua and Barbuda | AG | 17.06 | -61.80 | | Argentina | AR | -38.42 | -63.62 | | *:* | *:* | *:* | *:* | |
| *Data table* |  | *Location table* |

The Processing code used a for loop to go through each item in the *data* table (e.g. bad teeth), finding the each country name in each case. Using that country name, it then found the relevant row in the *location* table that corresponded to the relevant name. This allowed you to extract the correct latitude and longitude corresponding to each data item.

When you found your own datasets, it is likely that the number of rows in the two tables would be different to each other, if for example, not every country had an item of data associated with it. It was important therefore that when you used the for loop to count through each row, it counted the number of rows in the data table, not the location table. If you did not, you may have got the *Array out of bounds* error.

Another cause of problems I noticed with some people's sketches was a map where all circles appeared the same size or completely absent. A likely cause of this problem was an incorrect calculation of dataMin and dataMax from the data table. The section of code in the lecture notes that did this is shown below:

|  |
| --- |
| // Find the minimum and maximum values  for (int row=0; row<dataTable.getRowCount(); row++)  {    dataMin =min(dataMin,dataTable.getFloat(row,"NumBadTeeth"));    dataMax =max(dataMax,dataTable.getFloat(row,"NumBadTeeth"));  } |

This wasn't a problem with the code as such, but because the new data table you chose to load may contained some blank entries. When trying to read the blank entry and store the result as a number (getFloat(row,"NumBadTeeth")), Processing was having trouble working out what number a blank table cell should be. When it has trouble like this, Processing stores the results with a special code NaN (short for "Not a Number"). This caused problem when the sketch tried to work out the minimum and maximum data values.

To solve this particular problem you can do one of two things. The easiest, if your table isn't too large, is to edit the .tsv file and replace all the blank cells in the column you wish to read, with 0s. Alternatively, you can use the 'if' statement to read only non-blank cells from the table (more on 'if' in Session 3):

|  |
| --- |
| // Find the minimum and maximum values  for (int row=0; row<dataTable.getRowCount(); row++)  {    if (dataTable.getString(row,"NumBadTeeth").length() > 0)    {      dataMin =min(dataMin,dataTable.getFloat(row,"NumBadTeeth"));      dataMax =max(dataMax,dataTable.getFloat(row,"NumBadTeeth"));    }  } |

The extra line first reads in the table cell as a String (ie. as text). It then calls a method associated with Strings that counts its length (ie. how many characters in the text). Only if there are more than 0 characters (i.e. not blank) will it then try to extract the cell value as a number.

**Following the code examples in Ben Fry's *Visualizing Data* book**

Some of you may have started to follow the code examples in Ben Fry's Visualizing Data book that covered some similar material as that in this session. In particular, Chapter 3 introduces the idea of combining a map with two tables of data, one containing point locations, the other containing data values.

As I mentioned in the lecture, there have been a few changes to Processing since Ben wrote that book, and in particular, the [Table](http://processing.org/reference/Table.html) class built into Processing was not available when Ben wrote his book. Instead he provided his own Table class that behaved in a similar but not quite identical way. So, if you wish to recreate his examples from Chapter 3, you need to make a few minor changes to his code. Below is a summary of the changes required.

* On p.32 he suggests downloading the file he wrote Table.pde and dragging it into your sketch. There is no need to do this since it is built in to Processing.
* The code example on p.33 creates a new Table with the line

|  |
| --- |
| locationTable =new Table("locations.tsv"); |

* Instead, with the current version of Processing, you create and read in data values into a table with the line

|  |
| --- |
| locationTable =loadTable("locations.tsv","tsv"); |

* as discussed in the lecture notes (note how the file format of the file to read is specified here, allowing you to change file formats if your data were instead in, say, comma separated values (CSV) format). You should also replace the loading of the random.tsv file in a similar way.
* On p.36 where the sample code loads the data table, the old version in the book matches the data table with the location table with the following code:

|  |
| --- |
| String abbrev = dataTable.getRowName(row);  float x = locationTable.getFloat(abbrev, 1);  float y = locationTable.getFloat(abbrev, 2); |

* Instead, in your code, you should use the approach we covered in the lecture materials:

|  |
| --- |
| String abbrev = dataTable.getString(row,"columnName");  TableRow matchedRow = locationTable.findRow(abbrev,"columnName");  float x = matchedRow.getFloat("columnName1");  float y = matchedRow.getFloat("columnName2"); |

* There is no longer a method called getRowName() so we instead just get the String in column identified by*"columnName"* (you would change this to whatever the name is of the column you wish to be read). Additionally, we use Processing's own [TableRow](http://processing.org/reference/TableRow.html) to store the matched row, from which the two column values are extracted.

The remaining code can remain as it is, but just remember that in later examples that load data into a table, to adapt the relevant sections of code in a similar way.