**Programmng Workshop: Notes**

Some brief notes on Wednesday's extra programming class. It would be good if others who attended could add their own thoughts/contributions to the discussion board, especially for the benefit of those who could not make the session but wished to do so.

**Limbering Up**

We started with some simple exercises to get comfortable with getting Processing working, writing simple sketches and generating quickly, new versions of previously written sketches.

**"Write a very simple sketch to print a message in the console"**

|  |
| --- |
| // Hello world.    // Example of Processing code in 'static' mode: No setup() or draw().    println("Good morning everyone"); |

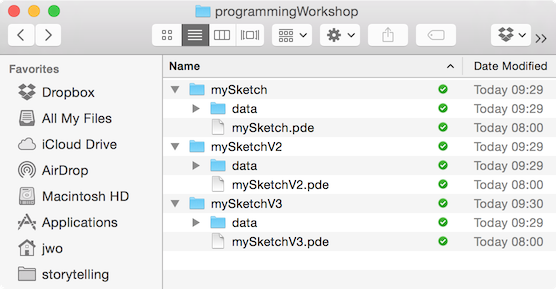
**"Write a sketch to display the numbers 0-9 in the console"**

|  |
| --- |
| // Displays the numbers 0-9. No drawing, just output to the console    void setup()  {    // A for loop for repeating something many times    for (int i=0; i<10; i++)    {      println(i);    }  }    void draw()  {    // This stuff happens lots of times (60x per second).    } |

**"Write a sketch that draws 10 circles"**

|  |
| --- |
| // Sketch to draw 10 circles in a line.  // Class of '16 group effort.    void setup()  {    size(900,400);   // Create a window 900x400 pixels.  }    // This stuff happens lots of times (60x per second).  void draw()  {    background(255);// Make the background white    fill(#5B7CAA);  // A lovely coral blue found from the colour selector in the tools menu.      // A for loop for repeating something many times    for (int i=0; i<10; i=i+1)    {      // To keep it simple, the loop counts from 0-9 but the ellipse x-position multiplies      // the i value by 80 to space the circles apart. This is probably simpler than using      // a different i value in the line that begins with 'for(int i=...'      ellipse(100+ i\*80,200,50,50);    }  } |

**Some useful tips when using Processing**

* Make use of the [Processing reference](https://processing.org/reference/) as you code. I use it all the time as I can never remember all the details of all the commands.
* If you double-click on any Processing command in you coding window then select Help->Find in reference Processing will bring up the relevant reference page.
* Make sure that in Processing->Preferences that you have both 'Continuously check for errors' and 'Code completion with Ctrl-space' ticked. This will ensure that you get red and yellow wavey lines under bits of your code that are likely to generate errors or warnings and also can be used to suggest what commands are available as you type (or press control-space).
* You can create different 'versions' of your sketches as you modify them simply by opening your file browser (Windows explorer or Mac Finder), going to wherever you have saved your sketch and copying the folder containing your sketch. You need to make sure the main sketch file inside that folder has the same name as the folder itself though. For example, if you have created a sketch called mySketch and you make a copy of it called mySketchV2, you would rename the file mySketch.pde in that second version to be called mySketchV2.pde  
    
  

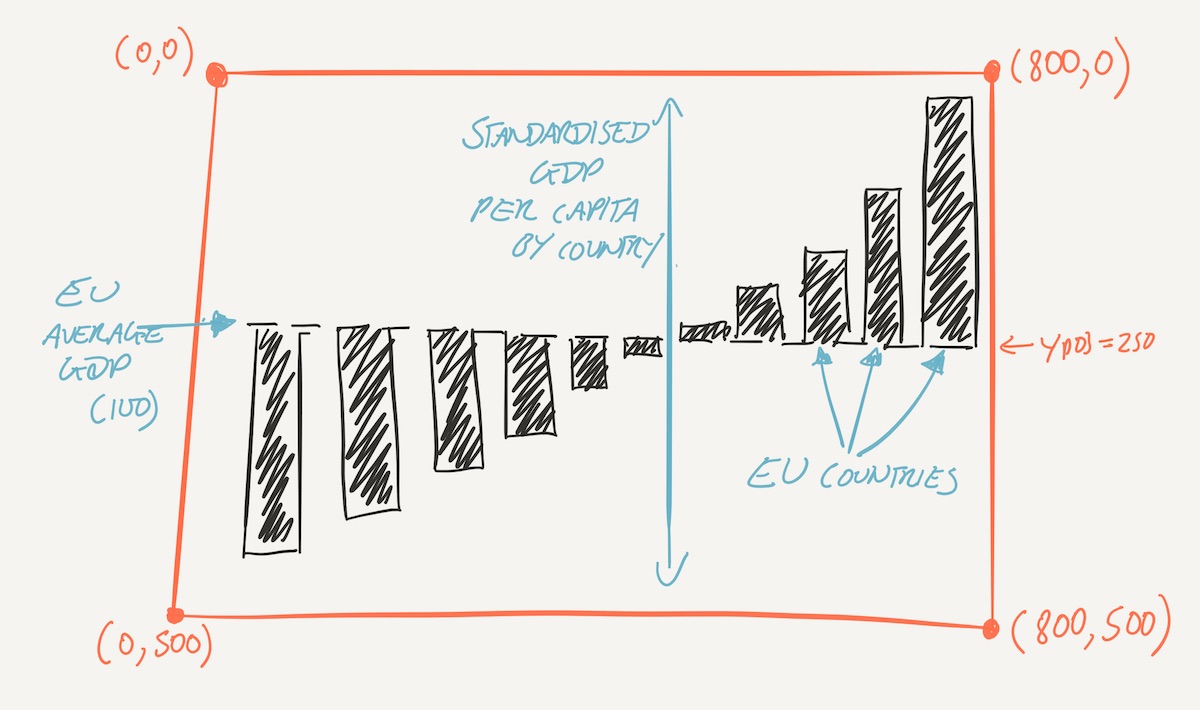
**Coding Tip:** *Start simple, grow your sketch step by step.*   
  
Don't try to write everything at once. Instead start with a very simple version of your sketch and make sure it behaves exactly as you expect it to. Then gradually add new functionality, at each stage ensuring it still works as expected before moving on to the next stage.

**European GDP Example**

We spent the second half of the morning working on designing a sketch to show some European GDP data. See [this table](http://gicentre.org/datavis/programmingWorkshop/euroTableSimpleFormatted.html) for the start point of the exercise.

As a group we looked at the data and then decided to create a sketch showing the last column representing standardised GDP values per person, expressed as a percentage of the European average GDP per person (a value of 100 means a country has the same GDP per person as the European average, a value of less than 100 meaning lower than the average, above being greater).

**Design Tip:** *Know your data.*   
  
Before you start sketching your design and certainly before you start coding, make sure you know your data, how it is structured, what it does and does not include. For example, we might wish to map the European GDP data, but looking at the table it becomes clear that there are no latitude or longitude (or other location coordinates) in the dataset, so mapping would be challenging.

Before we started any coding, we then discussed possible design choices, rejected some and then agreed on the following, which we sketched out:   
  


**Design Tip:** *Sketch your design on paper (or iPad or phone) before you code.*   
  
It is much easier to change your paper sketch, or reject it in favour of another than it is to code it up and then change it. A paper sketch will also help you when it comes to some of your coding and you need to think about what elements are drawn where.

**"Create a new template sketch that will read the data into some type of Processing variable"**

We recognised that the data were in tabular form, so the most obvious Processing variable type to store this is the [Table](http://processing.org/reference/Table.html).

|  |
| --- |
| // Start of sketch to display European GDP data.    Table gdpTable;   // Declare table variable for storing data.                     // This is at the top because we will need it                     // both in setup() and in draw().      void setup()  {    size(800,500);  // Set up the sketch area to reflect our design      gdpTable =loadTable("euroGDPSimple.tsv","header,tsv");      // As a first step, saving a web page of the table allows us to    // be assured we have read the data correctly into our sketch.    saveTable(gdpTable,"gdpTable.html");  }    void draw()  {    background(255);      // No drawing yet, just a white background.  } |

At this first stage, we added the line saveTable... naming a file with the .html extension that created a web page we could display and so check the data had been read correctly before continuing. Later versions of the sketch can remove that line, as it was there for checking only. The sketch above is an example of 'start simple, grow your sketch step by step'.

**Coding design tip:** *Coding achieves two things: (a) placing the bits of data you are interested into named variables that you can manipulate; (b) transforming those data into some graphic form.*   
  
So far, the code above is concerned with (a). We will need to go a little further with this to be able to extract individual elements we are interested in before we can turn them into the bars shown in the paper sketch.

**"Display the country names and the GDP values for each country in the console"**

Our next incremental step was to see if we could extract each country name (or its two letter 'NUTS\_code') place it into its own variable along with its associated GDP per capita value. Still no drawing, but a necessary stage:

|  |
| --- |
| // Next stage of sketch to display European GDP data.    Table gdpTable;   // Declare table variable for storing data.                     // This is at the top because we will need it                     // both in setup() and in draw().    void setup()  {    size(800,500);  // Set up the sketch area to reflect our design      gdpTable =loadTable("euroGDPSimple.tsv","header,tsv");      // No longer needed, so line below commented out.    //saveTable(gdpTable,"gdpTable.html");       // Ask processing to get ~all~ the rows. This is a 'repeated task' (once     // for each row in the table, so use a loop.    for(int i=0; i<gdpTable.getRowCount(); i++)    {      String country = gdpTable.getString(i,"NUTS\_code");      float gdp = gdpTable.getFloat(i,"GDPCapitaPPSPercEU");        println(country,gdp);    }  }    void draw()  {    background(255);      // No drawing yet, just a white background.  } |

We encountered various bugs along the way here, but because the sketch was still quite simple and we had previously confirmed we had been able to read in the table correctly, bugs were relatively easy to spot and correct.

By this stage we are able to store individual country names and their associated GDP per capita values in their own variables (country and gdp) so this puts us in a strong position to be able to draw them later.

It was pointed out in class that we were still missing a couple of other important bits of information necessary to implement our design - the smallest and largest GDP values in the dataset so we can scale the bars to use this full range.

**"Add code to calculate the minimum and maximum GDP values and display them in the console"**

We discussed this in some depth recognising a familiar pattern and making use of Processing's [min()](https://processing.org/reference/min_.html) and [max()](https://processing.org/reference/min_.html) functions (and a reminder to make frequent use of the Processing reference pages). We discussed why, when calculating the minimum value, we initialse the variable that will store this with a very large number, recognising that this might seem odd when it is a minimum we are interested in. The reason to initialise with MAX\_FLOAT is to ensure we do not inadvertantly record the minimum as whatever value we initialised our variable with (you can experiment by changing the MAX\_FLOAT below with 0 to see the effect).

|  |
| --- |
| // Start of sketch to display European GDP data.    Table gdpTable;   // Declare table variable for storing data.                     // This is at the top because we will need it                     // both in setup() and in draw().    float gdpMin,gdpMax;   // Declared here because we need these                          // values in both setup() and draw().  void setup()  {    size(800,500);    gdpMin =MAX\_FLOAT;    gdpMax =MIN\_FLOAT;      gdpTable =loadTable("euroGDPSimple.tsv","header,tsv");      // No longer needed, so line below commented out.    //saveTable(gdpTable,"gdpTable.html");      // Ask processing to get ~all~ the rows. This is a 'repeated task' (once    // for each row in the table, so use a loop.    for(int i=0; i<gdpTable.getRowCount(); i++)    {      String country = gdpTable.getString(i,"NUTS\_code");      float gdp = gdpTable.getFloat(i,"GDPCapitaPPSPercEU");        gdpMin =min(gdp,gdpMin);      gdpMax =max(gdp,gdpMax);      println(gdp,country);    }      println("Minimum:",gdpMin);    println("Maximum:",gdpMax);  }    void draw()  {    background(255);       // No drawing yet, just a white background.  } |

**"Allow the data to be displayed in GDP order"**

We recognised that our paper sketch design had the bars in GDP order, but the table had rows in alphabetical order of two letter country code. What is the best way of reordering the data? Java experts can do this within the code\*, but a simpler approach would be to load the .tsv file into a spreadsheet, then get the spreadsheet to sort the data by the relevant column before copying the newly sorted data back into the tsv file. No code has changed, but it will be much easier then to implement the design as planned.

\**Edited to add:* Since the workshop, I've discovered a really easy way of sorting tables programmatically from within Processing. There are a couple of methods sort() and sortReverse() that allow you to sort the table based on a given column index or column heading name. One thing to be careful of is that by default items are sorted alphabetically even if they appear to be numbers. This could result in an incorrect ordering of numeric data, for example 10,101,102,20,21,220. You can force a column to be treated numerically by setting its type with the setColumnType() method. So to order the GDP table ensuring the GDPCapitaPPSPercEU column values are in order from smallest to largest, you could include the following in setup():

|  |
| --- |
| gdpTable =loadTable("euroGDPSimple.tsv","header,tsv");  gdpTable.setColumnType("GDPCapitaPPSPercEU",Table.INT);      // Ensure this column is treated as integer values.  gdpTable.sort("GDPCapitaPPSPercEU");                         // Sort all rows so that data in this column go from smallest to largest. |

One of the big advantages of applying the sorting from within Processing, apart from saving some time, is that you could create a sketch that allowed the data to be resorted by the user depending on their interactions.

**Implementation Tip:** *Consider cleaning, sorting and arranging your data outside of Processing if you find it easier*   
  
How you do this would depend on what you need to get done (sorting in our example) and what platforms you are most comfortable. Some might prefer to use a spreadsheet, some some Python code, other a simple text editor, some the lines of code above. Just find a solution that works for you.

**Moving On**

This was as far as we got in the class, but we are now in a much stronger position to start working on the drawing part of the coding. We have a clear design sketch on paper. We have a clean and sorted dataset. We have found the full min-max range of the data. We know how to identify individual elements of the dataset inside of a loop. We also have from the earlier exercise a simple example of 10 circles laid out in a row, which is not dissimilar to what we need to do with our sketch, except we will be using rectangles not circles, and the height of those rectangles will depend on those GDP values and not a fixed size.

I would encourage everyone who attended the class (and anyone else who wishes to) to have a go at implementing the design and sharing their results on the discussion board. You should still follow the 'Start simple, grow your sketch step by step' approach, so the next step might be just to draw the 28 rectangles in a row each of the same height. Then try setting the height of each to the GDP value. Then try positioning them so they go above or below the average centre line depending on whether they are above or below 100.

If you manage this, here are some further activities that you could use to develop your design and coding skills:

* Add some annotation to the sketch such as a label for each bar, and a title. Positioning text is no different in principle to positioning any other graphical object such as a circle or rectangle. Have a look at the text related commands in the [Processing reference pages.](http://processing.org/reference)
* Add some simple interaction such as a tooltip that shows the absolute GDP figure (as opposed to the GDP per capita used to provide the bar height)
* Have a go doing something similar with this [more complicated table](http://gicentre.org/datavis/programmingWorkshop/euroTableFormatted.html) also showing the GDP data for EU member states. Unlike the simpler version, this one includes GDP values for regions within each member state. The data are hierarchical in that each row has a 'NUTS\_level' where 0 indicates data for the whole country (as used in the simple version of the table), 1 indicates larger regions within a country (e.g. London) and 2 indicates smaller regions within each larger region (e.g. Inner London East). Note that the 'NUTS\_code' for each row also reflects this hierarchy.   
    
  This is quite a complex table to process, but if you follow the 'start simple, grow you sketch step by step' approach you should be able to get something useful out of the data. There is, buried in the table a rather startling fact which a good data visualization would be able to bring out. See if you can find it!