**Session 3 - Representing data with colour**

This session shows how colour may be incorporated into the data visualization process. It introduces some of the important colour models and palettes and their representation in Processing. Guidance is given on what colour models and palettes are most appropriate for given data types.

**By the end of this session you should be able to:**

* produce a sketch that maps colour to data value
* convert between different colour spaces
* interpolate between colours when representing continuous data
* import the *giCentre utilities* library into your sketches
* incorporate Brewer colour schemes into your sketches

**Contents**

1. [3.1 Introduction](http://moodle.city.ac.uk/mod/page/view.php?id=580569#s3.1)
2. [3.2 Describing colour](http://moodle.city.ac.uk/mod/page/view.php?id=580569#s3.2)
3. [3.3 Mapping data to colour](http://moodle.city.ac.uk/mod/page/view.php?id=580569#s3.3)
4. [3.4 Conclusions](http://moodle.city.ac.uk/mod/page/view.php?id=580569#s3.4)

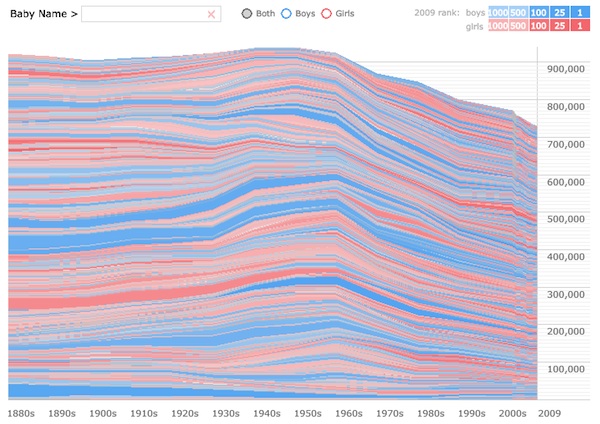
**3.1 Introduction**

We use colour in the representation of information on the web so ubiquitously that it is easy to forget the design decisions required to use it effectively. Colour is a complex topic that would merit an entire module by itself. This session will introduce some of the ways we can describe colour, how it may be most effectively associated with different data types and how to get Processing to do what we want with colour.

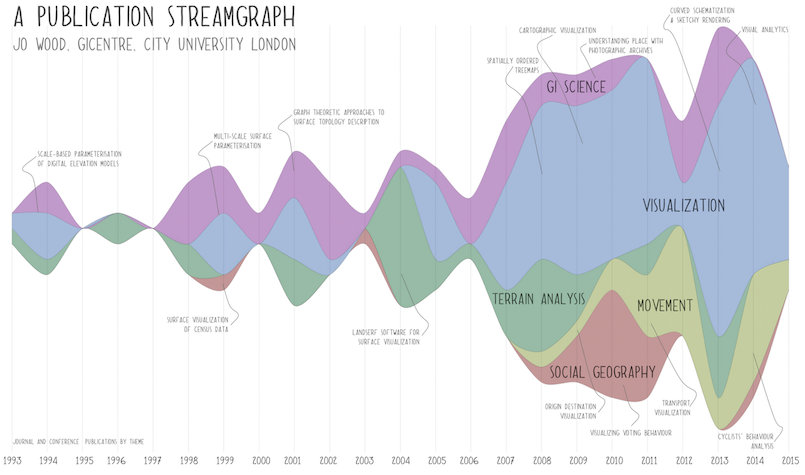
Before we look in detail at how we describe and use colour, let's consider an example. [Baby Name Voyager](http://www.babynamewizard.com/voyager) allows you to explore the popularity of names given to babies in the US over the last 130 years.

Explore the baby name voyager and see if you can find any interesting patterns. In what ways has colour been used to make finding these patterns easier?

The Baby Name Voyager is a simplified example of a 'stream graph' - a technique proposed by Lee Byron and Martin Wattenberg in 2008. It is a variation of a stacked bar chart that tries to maintain continuity of the horizontal strips. While it has some aesthetic appeal, it does suffer from the problem of different vertical orderings having a significant effect on the appearance of the chart. Lee Byron has provided the [Processing code to do this](http://www.leebyron.com/else/streamgraph/) should you be interested (suitable for more advanced programmers at this stage). Here is an example I created showing the [number and type of papers I have published over since 1993.](http://gicentre.org/datavis/session03/images/publicationStream.pdf)   
  
You may have also noticed that the graphic on the front page of this module is a stream graph, in that case of weekly attendance in class last year.

[](http://gicentre.org/datavis/session03/images/babyName.jpg)

Example from baby name voyager.

[](http://gicentre.org/datavis/session03/images/publicationStream.png)

Publication stream graph.

**3.2 Describing Colour**

If we wish to control the way in which we use colour in our data visualizations we need a precise way to describe the colours we use. While from a computational point of view, this is necessary, there is some interesting research that suggests that colour perception is linked to [gender](http://www.datapointed.net/2010/09/men-women-color-names/) as well as the language we use to describe colour.

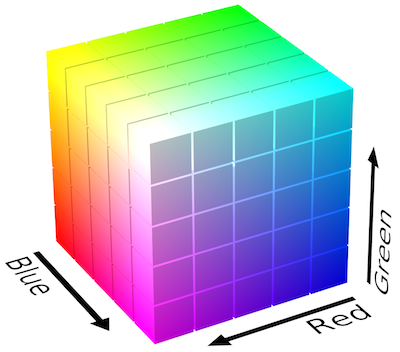
For commentary on this phenomenon, see the excellent [Eager Eyes blog](http://eagereyes.org/blog/2011/you-only-see-colors-you-can-name) by Robert Kosara.

[click to see sketch](http://gicentre.org/datavis/session03/rainbow/javascript/index.html)

Rather than subjective terms like 'dark red', 'pale orange' etc. we have already seen that in Processing we can describe colours with a set of RGB (red, green, blue) numbers. Scaling each number between 0-255, we can describe every colour the computer is a capable of generating, known as its [colour gamut](http://en.wikipedia.org/wiki/Gamut). The RGB system is common in computing because it maps well onto the way in which display screens generate coloured images. It is found across many computing languages and environments where specifying colours is required.

One of the problems with the RGB system though is that it may not be obvious without some practice, which numbers contribute to which colours. For example, while it may be obvious that to get a bright red colour we might use (255,0,0), what triplet is required for a dark brown or a pastel mauve? One way of conceiving of all possible reproducible colours is to think of a *colour space*. So for example, if we construct 3 chart axes at right angles representing the degree of red, green and blue components of a colour, we can plot an RGB colour cube.

This gives us some sense of how the three components can be combined to give us the colour we want (e.g. the cube shows that yellow is made up of high red and green components but a low blue component). Unfortunately many potential colours are not visible in a single view of the cube as they appear 'round the back' (e.g. browns), or 'in the middle' (e.g. greys). So while conceptually the RGB space has some simplicity, it does not map well to the way we conceive or see colour.

[](http://gicentre.org/datavis/session03/images/rgbCube.png)

RGB colour cube.

In order to make understanding the relationship between colour and RGB value easier, Processing provides a *colour selector* available from the Processing Tools menu (see below). This allows you to choose a colour interactively by moving a mouse over a coloured square and along a strip of different colour hues. As you drag the mouse, the R, G and B numbers on the right hand side update.

Use the Processing colour selector to find the RGB values of a dark brown and a pastel mauve. What colour corresponds to the RGB triplet (104,38,100)?

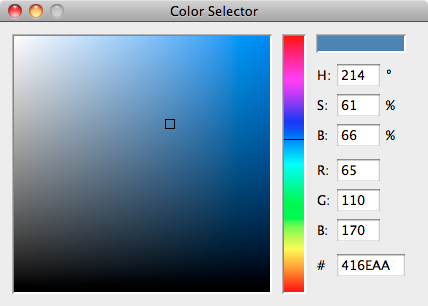
Brown is an unusual colour. When described using RGB values it is 'dark yellow'. Ware (2004, p.118) points out that unlike red, blue or green, to identify brown in a scene, we also need a 'reference white' for it to be perceived. He suggests that there is no such thing as a brown light in a darkened room (in contrast to, for example, a red, orange or green light). The idea that one colour can only exist with reference to another is an idea we will consider later in this session.

The colour selector uses an alternative space that is perceptually simpler to work with than RGB space *Hue*, *Saturation* and*Brightness* (HSB) is a 3d colour space that maps well on to our understanding of colour.

**Hue** is represented by the vertical strip of colours in the selector and corresponds to what we might loosely describe as "colour" in everyday language. It is what distinguishes blue from red, or purple from orange. Hue is usually represented as an angle between 0 and 360 degrees recognising that the sequence of colours 'wrap round' (the reds at the top of the hue strip in the selector are similar to the reds at the bottom).

**Brightness** (closely related to *lightness*, *value* or *intensity*) is as we might expect, what distinguishes dark from light shades and corresponds to the vertical axis of the large square in the Processing colour selector.

**Saturation** (closely related to *chroma*) describes how vivid a colour appears. It corresponds to the horizontal axis on the Processing colour square, with low values on the left giving a greyer washed out appearance and higher values on the right giving richer more vivid colours.

[](http://gicentre.org/datavis/session03/images/selector1.png)

Processing colour selector.

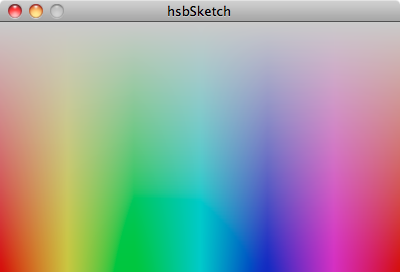
By default in Processing all colours are specified using RGB triplets, but you can easily change to using HSB colours using the [colorMode](http://processing.org/reference/colorMode_.html) command (note the American spelling). To see the HSB colour space being used consider the following sketch which plots a slice through the space at a fixed brightness.

|  |
| --- |
| // Draws a slice though HSB colour space  // Jo Wood, 26th January, 2016    float bri;   // Use to set colour brightness    void setup()  {    size(400,250);    bri = 75;   // Try changing this value between 0-100      colorMode(HSB,width,height,100); // Use HSB colour space.  }    void draw()  {    for (int x=0; x<width; x++)    {      for (int y=0; y<height; y++)      {          stroke(x,y, bri);          point(x,y);      }    }      noLoop();   // This stops sketch redrawing 60 times per second.  } |

Most of this sketch should be familiar to you. We create a numeric float variable called bri which will set the constant brightness of all colours shown in the sketch. This is initialised with the value of 75. In this example we set the colour mode to use HSB colours and additionally set the range of numbers used to describe that space. So the linecolorMode(HSB,width,height,100); allows hue to be specified as a number between 0 and width (400 in this example), saturation to be specified between 0 and height (250 in this example) and brightness to be specified between 0 and 100.

The draw() method uses two loops, one inside the other, to draw a [point()](http://processing.org/reference/point_.html) at each pixel coordinate in the cell. The colour of this point is set by the x,y position controlling hue and saturation and the constant bri controlling brightness.

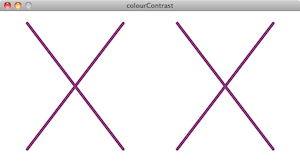
Finally, the [noLoop()](http://processing.org/reference/noLoop_.html) command tells processing not to redraw at 60 times a second, but instead, just draw the once. This can be useful for sketches that you know will not change in any way and can be useful for preserving battery life on any devices that run the sketch.

[](http://gicentre.org/datavis/session03/images/hsbSketch.png)

Output from HSB Sketch

So now we have the ability to specify colours as either RGB or HSB triplets, can we can now fully control the way our data are mapped to colours in our visualizations? Well, not quite, because as we shall see, colour *generation* does not always correspond to colour *perception*. To give you an idea of the role of colour perception, consider the sketch below that attempts to answer the question posed above: 'what colour is represented by the RGB triplet (104,38,100)?'.

|  |
| --- |
| // Draws a colour contrast figure 4.14 from Ware 2004.  // Jo Wood, 26th January, 2016.    boolean drawBackground;    void setup()  {    size(600, 300);    drawBackground =false;  }    void draw()  {    noStroke();      if (drawBackground)    {      // Red background.      background(150, 0, 0);        // Blue rectangles      fill(0, 0, 100);      rect(0, 0,width\*.08,height);      rect(width\*.42, 0,width\*.08,height);      rect(width\*.58, 0,width\*.35,height);    }    else    {      // White background.      background(255);    }      // Draw crosses    stroke(104, 38, 100);    strokeWeight(6);      line(width\*.09,height\*.08,width\*.41,height\*.92);    line(width\*.09,height\*.92,width\*.41,height\*.08);      line(width\*.59,height\*.08,width\*.91,height\*.92);    line(width\*.59,height\*.92,width\*.91,height\*.08);  }    void keyPressed()  {    if (key ==' ')    {      // Toggle coloured background variable.      drawBackground = !drawBackground;    }  } |

[](http://gicentre.org/datavis/session03/colourContrast/javascript/index.html)

Colour contrast sketch (click to activate). Compare the colours of the cross on the left and right. Now click somewhere on the sketch and then press the space bar. What colours are the crosses?

Colour contrast is an important aspect to consider when constructing data visualizations. Colours of items are not perceived in isolation, but in comparison with surrounding colours. You need to consider the wider context when thinking about how to incorporate colour into your data representation. We will see further examples of this in the classroom lecture.

A note about the Processing code. Most of this sketch uses familiar concepts from Processing, but there are a few new ones introduced here. Firstly we use a new type of variable boolean which instead of storing a decimal number (float), or whole number (int) or text (String), stores only the values true or false. This can be useful for representing two mutually exclusive states. In this case the variable indicates whether or not we should display the red/blue background.

We can get different sections of our sketch to run depending on whether a Boolean condition is either *true* or *false*. We do this with the if statement, and in this case also with the else that is paired with it. Note how the curly braces are used to identify the sections of the code that are dependent on the condition being either true or false.

There is one further Processing command for drawing here - [line()](http://processing.org/reference/line_.html) used for drawing lines between a pair of start and end coordinates. So we can now draw *ellipses* with ellipse, *rectangles* with rect, *points* with point and *lines* with line. The colour of all these features can be controlled with stroke() and fill().

Finally this sketch uses one of Processing's inbuilt interaction methods void keyPressed() to detect if a key has been pressed. Just like void draw(), this method is called repeatedly whenever the sketch is run. Inside the method we use the if statement to see if the space bar has been pressed. If it has, we change the value of drawBackground to false if it was previously true or true if it was previously false. This is done with the ! operator which is shorthand for 'not'.

We will look at keyboard handling like this in more depth in a later session when we consider how to incorporate interaction into your visualizations.

[click to see sketch](http://gicentre.org/datavis/session03/gridBoard/javascript/index.html)

**3.3 Mapping Data to Colour**

**Levels of Measurement**

To understand how to map data to colour, we need to be able to understand the type of data we are dealing with. A useful and widely used classification is that developed by Stevens (1946) that describes the *level of measurement* of a type of data. He classed data into two groups - *categories* of things and *measurements* of things. Categories include things like countries, hair colours, baby names, operating systems, and ranked orders (1st, 2nd, 3rd largest etc.). Measurements include things like distance, age, number of bad teeth, temperature. He further subdivided categories into *nominal*(unordered) and *ordinal* (ordered) groups and measurements into *interval* and *ratio* values.

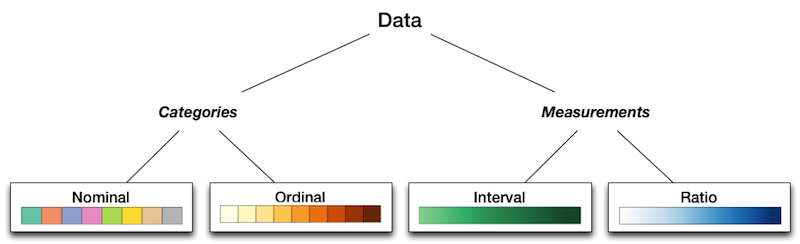
Munzner (2015) in her visualization analysis and design framework, uses a slightly different naming convention for data types. She divides them into *categorical* (equivalent to Stevens' *nominal* type), *ordinal* (matching Steven's ordinal type) and *quantitative* (combining both Stevens' *interval* and *ratio* types).   
  
But the impotant principle using either scheme is that it is necessary to ask what type of data we have before we can make the best design choice for colouring (and other visual marks and channels).

Why is this important? Because different levels of measurement are best suited to different colour schemes. The figure to the right summarises the relationship between level of measurement and colour scheme.

The colour schemes shown in the figure are only examples, but they have important characteristics that should apply to all colour mappings. Nominal data are by definition discrete categories of things that have no particular order. Therefore the colour scheme that reflects such data should also be discrete (i.e. have distinct differences between colours) and have no obvious order to them. If each data item in a nominal dataset is to have equal weighting in your graphic, the colour brightness (and to a large extent, the colour saturation) should be approximately equal between items. This leaves you with colour hue to distinguish between categories. That way, you *use your colouring to reflect the character of your data*.

Ordinal data are also categorical by definition, so the colour scheme should again be discrete. But this time we should use properties of colour to reflect the ordering present in the data. It should be obvious to the viewer of your colour scheme which colours represent 'more' than which others. Therefore hue, which is essentially unordered, will change very little, but brightness and/or saturation are likely to reflect the order of your data.

The same rules can be applied to the ordered measurement data, but this time since we don't have discrete categories of data, we do not need to use discrete colours. A *continuous* ordered colour scheme better reflects the continuity of measurement. The distinction between interval and ratio data is a slightly more subtle one. Interval data are measurement data that has no obvious 'start point'. Or to put it more formally, measurements where a value of 0 does not mean an absence of that measurement. Temperature on the Celsius or Fahrenheit scale is an example of this (0 degrees does not mean no temperature). To reflect this with colour, we don't want to give the colour scheme an obvious 'start point' such as pure black or pure white. In contrast, ratio scale measurements do have a fixed point at 0 indicating an absence of that measurement (e.g. distance or monetary value). In this case, including an obvious 'start colour' can make more sense.

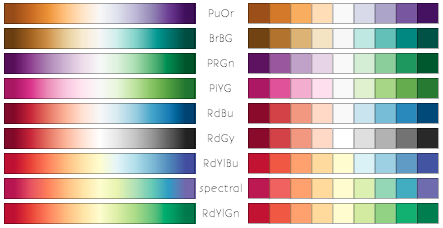
[](http://gicentre.org/datavis/session03/images/levelOfMeasurement.png)

Level of measurement (click to enlarge).

**Diverging Colour Schemes**

A common type of data that you may wish to visualize is the *diverging* dataset. This is a dataset that often shows change from some neutral point. For example changes in temperature around a seasonal average (could be negative if below average, positive if above). In such schemes we need to distinguish positive change from negative change and also the degree of divergence from the neutral centre point. This is done typically by using two hues to represent the two directions and increasing saturation or decreasing brightness away from the centre point. Some examples are shown below for both measurement (left) and categorical (right) diverging data.

**Aesthetics, Accessibility and Meaning**   
  
The choice of colour does of course have a big impact on the aesthetic appeal of the visualization you create. The rules above are important, but not the only consideration when choosing colours. You may wish to create a 'mood' for your graphics such as 'warm', 'vibrant', 'technical', 'informal' etc. The choice of colour palette can have a large influence on the feeling evoked by the reader of your visualization.   
  
There are some useful web sites that can help with the selection of colour palettes by sharing and organising user-contributed colour combinations. For example [Adobe Color CC](https://color.adobe.com/create/color-wheel), [paletton.com](http://paletton.com/) and [COLOURlovers](http://www.colourlovers.com/) which also provides a useful [colour palette creation tool](http://www.colourlovers.com/copaso/ColorPaletteSoftware).   
  
You should also consider that not everyone has the same ability to register and distinguish colours. Ensuring colour schemes are accessible to your audience, is important. [Contrast-A](http://www.dasplankton.de/ContrastA/) is a useful web-based software tool for selecting palettes of contrasting colours that are accessible to people who have some form of colour deficiency.   
  
Colour selection should also consider other meanings attached to colours that may be relevant to your data visualization. Colours are not always neutral in their association especially when associated with data about people. Red may be associated with blood, death, extreme heat, the political left (Europe), the political right (the US), debt (finance); yellow may be associated with prosperity (gold), cowardice, illness (hepatitis) etc. The language of racism and racial identity often makes use of colours as labels and can evoke strong feelings. A sensitivity to the context of your own data visualization and its audience is an important part of your design skillset.

[](http://gicentre.org/datavis/session03/images/diverging.png)

Some diverging colour schemes from Brewer (2003). Click to enlarge.

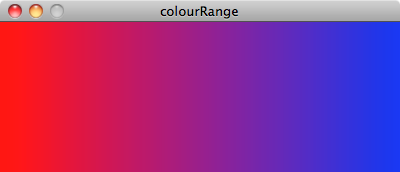
**Interpolating Colours in Processing**

So how do we go about creating ranges of colours in Processing? We have seen how to create individual colours using both RGB and HSB colour spaces, but to create entire ranges of colour we need to be able to *interpolate* between colours. Let's consider an example:

|  |
| --- |
| // Draws a simple colour range by interpolating colours.  // Jo Wood, 26th January, 2016    color colour1; // The two colours between which to  color colour2; // construct the colour range.    void setup()  {    size(400,150);    colour1 =color(255,0,0);    colour2 =color(0,0,255);  }    void draw()  {    for (float x=0; x<width; x++)    {      color newColour =lerpColor(colour1,colour2,x/width);      stroke(newColour);      line(x,0,x,height);    }      noLoop();   // Draw once only.  } |

Here we declare two new variables of type color (rather than float, boolean etc.). The variables are intialised in setup() to hold a red (colour1) and blue (colour2) colour. In the draw() method we use a for loop to draw a series of vertical lines from the left to the right of the sketch. The colour of each line is set by the third variable newColour.

newColour is *interpolated* between the red and the blue using the Processing command [lerpColor()](http://processing.org/reference/lerpColor_.html). This method requires three parameters in the brackets. The first two represent the start and end colours (red and blue in this example). The third is a number between 0-1 where the closer to 0, the nearer the interpolated colour will be to the first colour and the nearer to 1, the closer it will be to the second colour. In this example that number is generated by dividing the value of x by the width of the sketch.

[](http://gicentre.org/datavis/session03/images/colourRange1.png)

Output from colour interplation sketch (click to enlarge);

Perhaps a more useful example to consider is one that maps an interpolated colour scheme onto some real data. The example below shows how we might use lerpColor() to symbolise temperatures over a 5 day weather forecast. Because temperature is measured on a continuous interval scale, we can use saturation and brightness to vary temperature magnitude. Because of the association of cold temperatures with blue and warm ones with red, we can also afford to vary hue slightly, although not by too much.

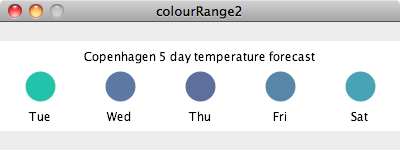
|  |
| --- |
| // Shows the Copenhagen forcast temperatures in the next 5 days.  // Jo Wood, 26th January 2016    float tue,wed,thu,fri,sat; // Daily max temperatures.  color minCol,maxCol;       // Colours representing temp range.    void setup()  {    size(400,90);    noStroke();    textAlign(CENTER);      // Set the forecast temperatures and range    tue = 4;    wed = -2;    thu = -3;    fri = -1;    sat = 1;      // Use HSB colour space.    colorMode(HSB,360,100,100);    minCol =color(238,45,55);  // Blue in HSB space.    maxCol =color(165,67,72);  // Pale green in HSB space.  }    void draw()  {    background(0,0,255);      fill(lerpColor(minCol,maxCol,norm(tue,-4,4)));    ellipse(width\*.1,height/2,30,30);      fill(lerpColor(minCol,maxCol,norm(wed,-4,4)));    ellipse(width\*.3,height/2,30,30);      fill(lerpColor(minCol,maxCol,norm(thu,-4,4)));    ellipse(width\*.5,height/2,30,30);      fill(lerpColor(minCol,maxCol,norm(fri,-4,4)));    ellipse(width\*.7,height/2,30,30);      fill(lerpColor(minCol,maxCol,norm(sat,-4,4)));    ellipse(width\*.9,height/2,30,30);      // Draw text labels.    fill(0,0,0);    text("Copenhagen 5 day temperature forecast",width\*.5,20);    text("Tue",width\*.1,height-10);    text("Wed",width\*.3,height-10);    text("Thu",width\*.5,height-10);    text("Fri",width\*.7,height-10);    text("Sat",width\*.9,height-10);      noLoop();   // Draw once only.  } |

We declare the five float variables to store the 5 daily maximum temperatures at the top of the sketch since they will be used by both setup() and draw(). We initialise their values (i.e. the five temperatures) in setup() since this is a one-off process.

For this colour range we use HSB space by setting colorMode() appropriately. We initialise two color variables with the blue representing the minimum temperature (minCol) and a pale green representing the maximum temperature (maxCol). Both of these colours are defined in HSB space where hue varies between 0-360, and saturation and brightness vary between 0-100. The colours themselves were chosen using Processing's Colour Selector.

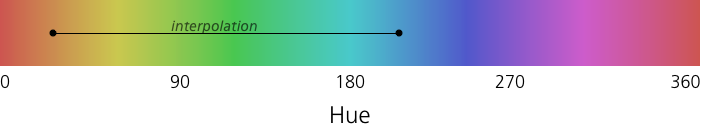
The draw method draws five circles representing the daily temperatures each coloured according to the interpolated colour between -4 degrees C representing the coldest colour and 4 degrees C representing the warmest colour. This is done with lerpColor() as we have seen before, but to scale each temperature between 0-1, we use the Processing function[norm()](http://processing.org/reference/norm_.html) that does the scaling to a standardised 0 to 1 range for us.

Finally we set the fill colour to black and draw some text labels using the Processing command [text()](http://processing.org/reference/text_.html). This simply displays some text at the given (x,y) coordinates. Text is centred at the given coordinates because setup() also included the command [textAlign(CENTER)](http://processing.org/reference/textAlign_.html).

[](http://gicentre.org/datavis/session03/images/colourRange2.png)

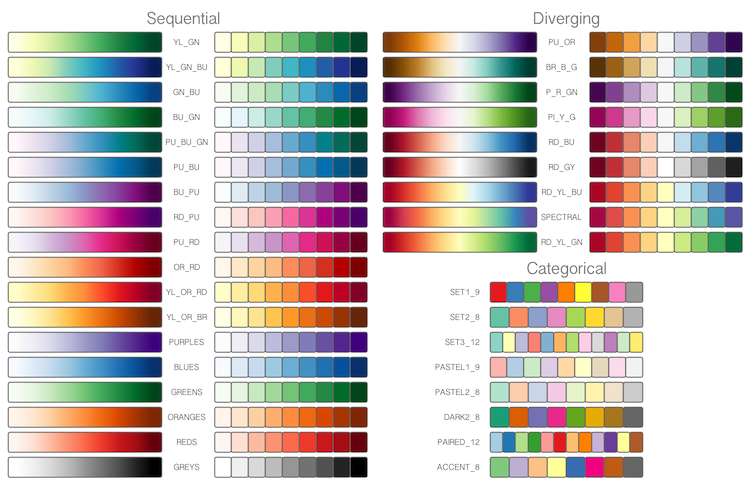
Output from Copenhagen sketch (click to enlarge);

**Preset Colour tables**

You may have noted in the temperature example that our range of colours is not ideal. While we may wish blue to represent the coldest temperature and perhaps orange to represent the warmest colour, if we interpolate hues between these two colours in HSB space we end up creating some undesirable intermediate green colours:   
  


What may be preferable is to create a customised colour range. To do this we have to do a little more work. Rather than try to find the RGB or HSB colours representing all the colours we wish to interpolate between, we can rely on the work of others who have defined a set useful *colour tables*.

ColorBrewer ([colorbrewer2.org](http://colorbrewer2.org/)) developed by Brewer et al (2003) provides a set of useful colour tables designed originally for mapping data. The palettes allow you to select sequential, diverging and categorical colour schemes depending on data type. You can use colorBrewer colours in your own Processing sketches by importing a *library* into your sketch that contains the necessary colour information. In this case we shall use the [giCentre Utils library](http://www.gicentre.org/utils/). Before we look at how to import the library, to the right is the range of colour palettes available to you. Each one has a name defined by the hues used (for single hue palettes), abbreviated colours at the end points (for two hue palettes) and abbreviated colours at the ends and mid points (for three hue palettes). The unordered categorical palettes each have a unique name followed by a number corresponding to the number of categories.

[](http://gicentre.org/datavis/session03/images/brewer.png)

ColorBrewer Palettes displayed by giCentre Utilities (click to enlarge).

To import the giCentre Utils library into your sketch, select Import Library->Add Library from Processing's Sketchmenu. From the list of libraries provided, select giCentreUtils (you may wish to filter by this name to make finding the library easier) and install. You can now include code that uses this library in your sketch. Below is a simple example that colours some data using the OrRd (Orange-Red) colour palette:

|  |
| --- |
| import org.gicentre.utils.colour.\*;   // For colour tables.    // Sketch to show surface temperature changes using a Brewer colour scheme.  // Jo Wood, 26th January, 2016    float temp1960,temp1970,temp1980,temp1990,temp2000,temp2010;    ColourTable cTable;   // Will store a Brewer colour table.    void setup()  {    size(400,150);    stroke(180);    textAlign(CENTER);      cTable = ColourTable.getPresetColourTable(ColourTable.OR\_RD,0,1);      temp1960 = 0.06;    temp1970 =-0.04;    temp1980 = 0.18;    temp1990 = 0.34;    temp2000 = 0.55;    temp2010 = 0.70;  }    void draw()  {    background(255);      fill(cTable.findColour(temp1960));    rect(width\*.05,50,width/8,height/3);      fill(cTable.findColour(temp1970));    rect(width\*.20,50,width/8,height/3);      fill(cTable.findColour(temp1980));    rect(width\*.35,50,width/8,height/3);      fill(cTable.findColour(temp1990));    rect(width\*.50,50,width/8,height/3);      fill(cTable.findColour(temp2000));    rect(width\*.65,50,width/8,height/3);      fill(cTable.findColour(temp2010));    rect(width\*.80,50,width/8,height/3);      fill(0);    text("Global surface air temperature anomaly",width/2,20);    text("5-year mean, base 1951-1980. Source: NASA, 2010",width/2,40);    text("1960",width\*.11,height-30);    text("1970",width\*.26,height-30);    text("1980",width\*.41,height-30);    text("1990",width\*.56,height-30);    text("2000",width\*.71,height-30);    text("2010",width\*.86,height-30);      noLoop();     // Draw once only.  } |

The form of this sketch is similar to the Copenhagen forecast except this time we use a ColourTable to store and calculate the colours associated with each temperature value rather than lerpColor().

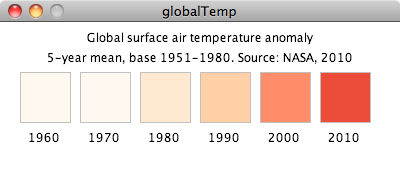
When using external libraries, we need to tell Processing that we are using external code, so the first line of the sketch imports the relevant colour table code.

The colour table is initialised in setup() with the line ColourTable.getPresetColourTable(). This uses the OR\_RDBrewer scheme and scales the colours between 0 and 1.

The draw() code draws a series of rectangles spaced across the sketch as a fixed proportion of the sketch's width and height. The colour of each rectangle is determined by filling it with the colour extracted from the colour table based on the values temp1960, temp1970 etc.

The final part of the sketch simply draws the text in black, much as we saw in the 5-day forecast sketch.

You can find out more about how to use giCentre Utils to create colours in your sketches by looking at the [giCentreUtils colour documentation](http://www.gicentre.net/utils/colour).

[](http://gicentre.org/datavis/session03/images/globalTemp.png)

Output from global temperatures sketch (click to enlarge).

**3.4 Conclusions.**

This session has considered how to design the selection of colours in your data visualization to match the types of data you wish to represent. While specifying colour precisely with RGB triplets is easy, selecting the appropriate colour for a given visualization task is more complex. The nature of the data you are representing will determine in part how to choose an appropriate colour, but other effects such as local context and purpose of the visualization will also have an influence.

We have seen how Processing may be used to specify colours in RGB and HSB space and how it may be used to interpolate colour ranges from colour end points. For more perceptually uniform colour ranges, the ColorBrewer schemes offer a useful alternative. These can be used in your sketches by importing the giCentre Utilities library. As we shall see in future weeks, one of the powerful features of Processing is the ability to incorporate external libraries into your sketches.

**References**

**Brewer, C., Hatchard, G. and Harrower, M.** (2003) ColorBrewer in print: A catalog of color schemes for maps. *Cartography and Geographic Information Science*, 30(1), pp.5-32

**Stevens, S.** (1946) On the theory of scales of measurement. *Science 103(2684) pp.677-680*.

**Ware, C.** (2004), Ch.4 Colour, pp.97-144 in *Information Visualization: Perception for Design*, London: Morgan-Kaufmann

**Recommended Reading**

*You are advised to look at at least one of the following:*

*Chapter 10 of Munzner (2015) contains a thorough discussion of colour, both in relation to perceptual issues and data type.*

**Munzner, T.** (2015) Chapter 10: Map Color and Other Channels, pp.219-241 in *Visualization Analysis and Design*, CRC Press

*The second half of Chapters 3 in Fry (2008) includes a discussion on mapping different colour schemes to data. The Shiffman tutorial provides an easy introduction to using colour with Processing and the Stone paper gives more general advice on using colour with data visualization.*

**Fry, B.** (2008) Chapter 3: Mapping, pp.37-42 in *Visualizing Data*, O'Reilly

**Shiffman, D.** (2010) Processing Tutorial: Color, [processing.org/learning/color](http://processing.org/learning/color/)

**Stone, M.** (2006) Choosing Colors for Data Visualization, [www.perceptualedge.com/articles/b-eye/choosing\_colors.pdf](http://www.perceptualedge.com/articles/b-eye/choosing_colors.pdf)