**Session 1 - Building data visualization applications**

This session is designed to introduce the process of building applications for data visualization. It introduces the Java-like software *Processing* and some data visualization models that help to organise the steps required to produce effective data visualization applications.

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

* install Processing, use the Processing Development Environment (PDE) and refer to the documentation
* organise sketches into a structured sketchbook
* build a simple sketch in the Processing Development Environment representing some data
* build simple sketch to demonstrate interaction
* use Munzner's *What-Why-How* framework to structure your visualization design decisions
* break down a data visualization task into Fry's 7-stage workflow

**Contents**

1. [1.1 Introduction](http://moodle.city.ac.uk/mod/page/view.php?id=578877#s1.1)
2. [1.2 What is data visualization?](http://moodle.city.ac.uk/mod/page/view.php?id=578877#s1.2)
3. [1.3 Introduction to Processing](http://moodle.city.ac.uk/mod/page/view.php?id=578877#s1.3)
4. [1.4 Data visualization models](http://moodle.city.ac.uk/mod/page/view.php?id=578877#s1.4)
5. [1.5 Conclusions](http://moodle.city.ac.uk/mod/page/view.php?id=578877#s1.5)

**1.1 Introduction**

Welcome to your first session in Data Visualization, which I hope you find interesting and useful. Before starting this session, you should have read the 'Module Information' section on Moodle, which outlines the aims, objectives and structure of this entire module.

This and subsequent sessions contain mini tasks and questions that will help you to explore some of the important ideas in data visualization. To help keep these notes readable, questions will be identified as shown below. This module is designed to be practical and requires active learning so please take the questions seriously and take time to answer them.

Questions and mini-tasks look like this. You should normally complete them before continuing with the rest of the lecture.   
  
Sometimes, to encourage you to pause and think about a question, you will need to click on a button such as the one below:   
  
*Click to reveal/hide answer.*

At various points, you will also find *asides* - pieces of background information or small self-contained sections that develop an idea in greater depth. These are separated from the main text so that they do not interrupt the main flow of the lecture text.

Asides will look like this. You can choose to ignore them, but reading them may provide you with interesting topics of conversation with friends and family.

This module will be making extensive use of the visualization software [Processing](http://processing.org/). This is the software that will allow you to create interactive graphics that can be shared on the web. You should install a copy of the latest stable version of Processing (version 3.0.1 at the time of writing) on your own computer as you will be creating many data visualizations with it over the coming weeks. These lecture notes will include plenty of examples of Processing code to show you how to implement these graphics.

// Draws an ellipse that follows the mouse position.

// Jo Wood, 10th January, 2015

void setup()

{

size(400,250);

}

void draw()

{

// Draw transparent background.

fill(255,10);

rect(0,0,width,height);

// Draw red ellipse at mouse position.

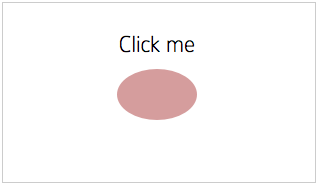
fill(180,120,120);

ellipse(mouseX,mouseY,70,50);

}

Processing code that produces some data visualization is known as a *sketch* and will be displayed as in the example above. To transfer any of the code to your own sketch, simply select it with the mouse and copy and paste into Processing (more on this later).

In many cases, these notes will also include a working version of the sketch. This will usually be shown as a clickable image. Try clicking on the picture above right to see what the sketch above does.

[](http://staff.city.ac.uk/~jwo/datavis/session01/clickme/javascript/index.html)

Output from the sketch (click to activate).

**1.2 What is Data Visualization?**

Data visualization is a recent discipline to emerge from the need to analyse and communicate patterns in complex datasets. While there is a long history of using pictures and graphics to represent data, we can consider modern data visualization as often having a number of identifiable characteristics:

1. Represents a complex dataset graphically that could not as effectively be represented via other means.
2. Frequently involves *interaction* to allow a user to control what they see.
3. Often emphasises *connections* and *comparisons* between items of data.
4. Is a "*kind of narrative providing a clear answer to a question without extraneous details*" (Fry, 2008 p.4).
5. Designed with an aesthetic appeal in order to encourage users to engage with the data or question.

So, does this barchart, showing the numbers of people enrolled on this module and the courses they come from, meet any of our criteria for data visualization?

It certainly represents a dataset graphically, although in this example, it is not a particularly complex one. There is some interaction involved in generating the graph (in the classroom we will interact with the visualization), but very little interaction in querying it. Does it allow connections or comparisons to be made? The use of colour does allow some simple comparisons between undergraduate and postgraduate numbers as well as the obvious comparison between numbers from different courses. Is there a narrative in the visualization? Possibly, if we are interested in conveying where the collective experience of everyone on the module derives, perhaps this contributes. Is there an aesthetic appeal that encourages you to engage with the data? What aspects of the design appeal and what do not?

Consider some data visualizations of more complex datasets. The 2015 Information is Beautiful awards showcase [some examples of data visualization](http://www.informationisbeautifulawards.com/showcase?acategory=data-visualization&action=index&award=2015&controller=showcase&page=1&pcategory=long-list&type=awards). The [Hubway Data Visualization Challenge](http://hubwaydatachallenge.org/) shows several examples of data visualizations that all address the same problem.   
  
Using the 5 criteria listed above, select what you think represents a good example of data visualization from one of those web sites. Jot down a brief justification of your choice for each of the 5 criteria. Are there any examples in that list that clearly fail as data visualizations?

For an overview of how data visualization has been used to reveal important patterns in data, you should devote an hour to watching the online documentary [Journalism in the Age of Data](http://datajournalism.stanford.edu/) that includes talks with some of the key players in the development of modern data visualization. While the focus is on journalistic approaches to data visualization, it emphasises the underlying approach of exploring data visually and communicating it effectively with modern technologies.

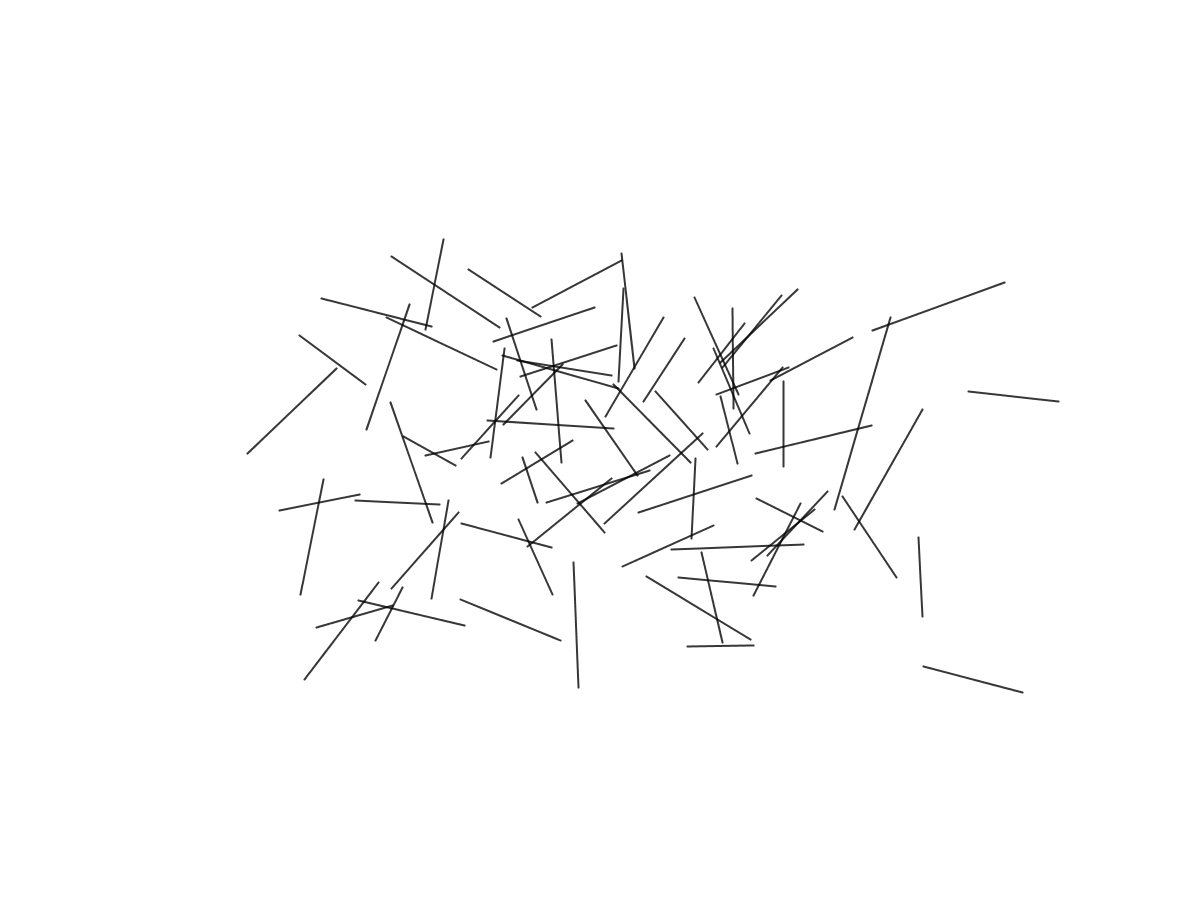
In her book *Visualization Analysis and Design*, Tamara Munzner asks an important question — why do data visualization (Munzner, 2015, pp.1-9)? This is important because in answering it, you should find it easier to design effective data visualizations. She considers a 'decision-making loop' where the visualization helps you to decide something, perhaps where to invest resources, what further information to gather etc. She considers the 'computer in the loop' and the 'human in the loop'. In other words, what is it that computation can offer in supporting decision making and what is it that humans can offer.

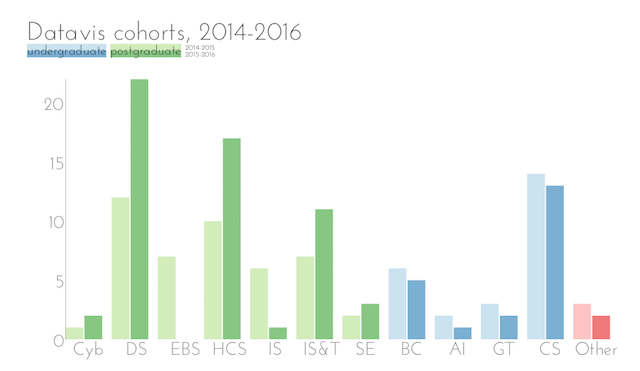
As an example imagine you have a dataset that represents reported crime in various districts in a region. If you simply wanted the answer to the question "which district has the highest reported crime?", data visualization would probably not be very helpful here because you could answer the question with a simple database query (i.e. it is the computer in the loop). If on the other hand a crime analyst wanted to know whether reported crime reflected similar patterns and trends to perceived crime patterns or they wanted to expore the data to see if there were any unexpected patterns or outliers that might suggest a change in policy, visualization might be more useful (i.e. decsion-making that requires the human in the loop).

If you design a data visualization to answer a question such as 'what is characteristic x in my dataset?' (e.g. which country has the highest GDP?), data visualization will have little to offer that you couldn't find out more easily with a simple database query. Good data visualization answers more complex questions that don't have an easily computable answer.

Over the coming weeks, we well refine our critical ability to evaluate good design in data visualization. We will develop rules and heuristics that will help in effective design, from the use of colour and symbolisation to layout and interaction. This should help you both to evaluate others' data visualizations, and to design your own more effectively. But first, let's consider how we go about building a data visualization application.

**1.2.1 What do we see when we visualize data?**

Imagine we have some quantity represented by these 80 lines — the length of each line represents the magnitude of some variable. Spend 30 seconds looking at the image below and consider what it tells you about the data the lines represent.   
  
In class we will spend some time asking questions about the data represented by these lines and consider how visualization design and understanding of human perception and cognition can help us help us design more effectively.

[](http://staff.city.ac.uk/~jwo/datavis/session01/images/cohort2016.png)

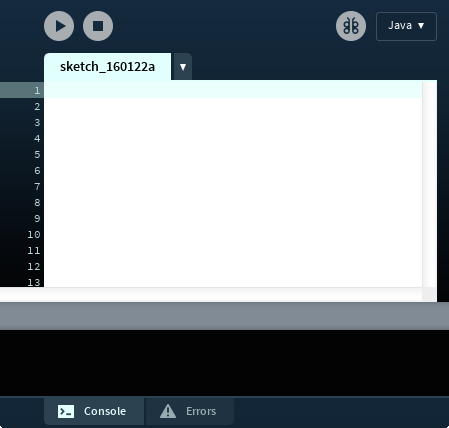
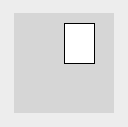
Simple data visualization (click to enlarge).

**1.3 Introduction to Processing**

[Processing](http://processing.org/) is the software we will be using throughout this module to create data visualizations. The module is practically focussed - you will be creating many examples with Processing each week to explore different aspects of the data visualization design process. Processing is available for PCs, Macs and Linux operating systems and can be used to produce stand-alone applications as well as javascript-based visualizations for the web and Android apps that can take advantage of mobile device functionality.

Processing is based around the [Java programming language](http://java.com/), but has been specifically designed so that people without programming experience can create applications quickly and easily.

Processing has its origins in the digital visual arts and has been designed to allow people to explore visual creativity while minimising the barriers that some programming languages present to those not confident computer programming. To see its context in the visual arts, you may wish to spend some time watching the documentary *Hello World! Processing* 

Now might be a good time get Processing installed on your computer. Go to [processing.org/download](http://processing.org/download/) to download a copy of the software, **Version 3.0.1** (or later stable release), for your computer. After installation, start up Processing to check it is working as expected. You should see a window appear similar to the following:   
  
   
  
Commands to issue drawing instructions are typed in the main window and the program can be run by clicking on the triangle towards the top left corner.   
  
To check Processing is working, try typing   
  
rect(50,10,30,40);   
  
and then pressing the 'Run' button. This should produce a small window as follows:   
  
   
  
The command you have just typed in draws a rectangle 50 pixels from the left of the window, 10 pixels down from the top with a width of 30 pixels and height of 40 pixels.

Instructions in processing are *case sensitive* and are generally separated with semicolons. Each line is likely to be a mixture of Processing *commands* (rect in the example above), *parameters* (the numbers 50,10,30 and 40 in the example above),*control structures* (discussed below) and *variables* (more on these below). Processing will colour each of these differently to help make the code you write easier to read.

Processing provides extensive documentation and tutorials to help you learn the language. In particular, you should probably bookmark the following:

* [Processing reference](http://processing.org/reference/), which lists and describes all of Processing's commands.
* [Processing tutorials](http://processing.org/learning/) on a range of topics from getting started to advanced image manipulation.
* [Processing examples](http://processing.org/learning/topics/) organised by topic.

These are also all available from the Help menu of Processing itself.

For those who have programmed in the Java language, you will find the syntax of Processing very familiar. In fact, Processing is simply a collection of Java libraries, an Integrated Development Environment (IDE) and simple Java pre-processor. It is compatible with standard Java and can be integrated with other Java libraries and classes. It is possible to create Processing applications with other IDEs such as BlueJ or Eclipse, but for this module, we will stick to using Processing's own IDE.   
  
If you have never programmed before and feel this is all a bit daunting, don't worry! The skills you need to get Processing to produce the kinds of data visualization you want will come with practice. To help getting started you may wish to work through the [interactive video tutorial by Daniel Shiffman](http://hello.processing.org/).

**1.3.1 The Sketch**

Central to the design of Processing is that it encourages quick experimentation with graphical design options. A program consisting of a set of Processing instructions is known as a *Sketch* to help emphasise the quick design experimentation flavour of the language. You can save a sketch to your computer easily by pressing the Save button (the downward pointing arrow button, or CTRL-S / cmd-S). This will save the code and any data associated with it in its own folder. It is a good idea to keep your sketches together in a 'sketchbook' location, giving each sketch a sensible name that reflects its purpose.

Consider the structure of a typical sketch as shown below. You can copy this code by selecting it with the mouse and pasting it directly into a new Processing sketch.

// Draws an ellipse that follows the mouse position.

// Jo Wood, 25th January, 2016

void setup()

{

size(400,250);

}

void draw()

{

// Draw transparent background.

fill(255,10);

rect(0,0,width,height);

// Draw red ellipse at mouse position.

fill(180,120,120);

ellipse(mouseX,mouseY,70,50);

}

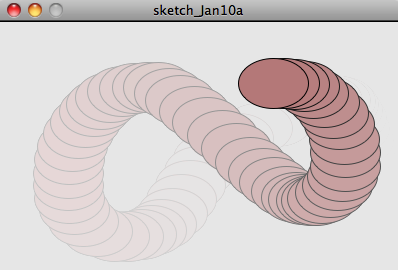
Most sketches will show the same basic structure as illustrated here.   
  
There are several lines of **commands**, each separated by a semicolon.   
  
Lines that start with a // are ***comment lines*** and are ignored by Processing, but they can be useful to remind you (and others) what your sketch does.   
  
Sketches usually have at least two **methods** - these are blocks of code enclosed by curly braces that perform some specific purpose. The two methods that all your sketches are likely to have are the setup() method where you provide the instructions to get your sketch started, and the draw() method which contains the instructions that are repeatedly called when the sketch is run.   
  
By default, any code inside the draw() method will be called repeatedly, up to 60 times a second. It is this that allows you to add animation and interaction to your sketches. It is important to remember that only code that actually does drawing should be placed in the draw() method; code that, for example, loads some data from the web would be done in the setup() method since this only need be done once, and not repeatedly called many times a second.

The first line of setup() should always be the size() command, where you tell Processing how large you wish your sketch to appear. The two numbers correspond to the width and height of the sketch in pixels. In the example above therefore, this line will create a sketch window that is 400 pixels wide by 250 tall.

The appearance of drawn features is controlled largely by two aspects - the **stroke** and the **fill**. The stroke is the line that surrounds most drawn features and the fill is the interior colour of the feature. By default, the stroke colour is black and the fill colour is white. These can be redefined with the stroke() and fill() commands. Colours are defined using one or more numbers each scaled between 0-255, the number of parameters determines how the colour is chosen:

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of parameters** | **Meaning** | **Example** | **Colour** |
| 1 | Greyscale | fill(0); |  |
| 1 | Greyscale | fill(255); |  |
| 1 | Greyscale | fill(128); |  |
| 2 | Greyscale with transparency | fill(0,30); |  |
| 3 | Red, Green, Blue | fill(200,128,64); |  |
| 4 | Red, Green, Blue, transparency | fill(200,128,64,10); |  |

In the sketch example above, a rectangle is drawn with a transparent white interior - fill(255,10); and an ellipse is drawn with a redish interior - fill(180,120,120);.

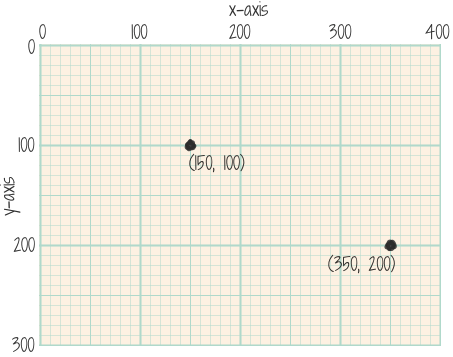
[](http://staff.city.ac.uk/~jwo/datavis/session01/clickme/javascript/index.html)

Moving ellipse sketch (click to activate).

The drawing of features in the example is done with the rect() command for drawing a large rectangle and the ellipse() command for drawing (you've guessed it) the ellipse. The rect() command takes four parameters in the order (x,y,w,h) where x and y are the pixel coordinates of the top left of the rectangle's position and w and h are its width and height. Likewise ellipse() also takes four parameters (x,y,w,h) where x and y are the centre of the ellipse and w and h are its width and height.

The only other new element of the sketch is the use of *variables*. These are named entities that can store a number of some kind. In this example we make use of 4 special variables. width and height will always store the width and height of the sketch window in pixel units. This can be useful when you wish to scale your drawing to use the full (or fixed proportion of the) size of the window. So in the example sketch, the second command in draw() draws a rectangle from (0,0) [top-left] to (width,height) [bottom-right]. This effectively clears the background of the sketch, in this case, with a semi-transparent white colour.

The mouseX and mouseY variables will always contain the current position of the mouse relative to the top-left corner of the sketch. By drawing an ellipse at these mouse coordinates, we can make it move around the screen in response to mouse movements. Remember, the draw() method is automatically called 60 frames a second, so as we move the mouse, the sketch will update. Because we have redrawn a transparent rectangle each time the method is called, the effect is to produce an afterimage-like effect.

[](http://staff.city.ac.uk/~jwo/datavis/session01/images/coordinates.png)

Processing's coordinate system for a sketch of size (400,300) with two example point locations specified as (x,y) pairs (click to enlarge).

**1.3.2 Simple Variables**

The ellipse example above made use of two sets of variables that are built in to Processing - screen dimensions (widthand height) and mouse position (mouseX and mouseY). Such built in variables are indicated in pink when viewing code in Processing.

We can get Processing to do much more powerful things when we create our own variables. Creating a variable involves declaring a *type*, a *name* and a *value*. For the moment, we can regard all variables that store a number as having a type float. We then just have to give our variable a name and a value. The name should always start with lower-case letters and not contain any spaces or special characters. For example,

float cost = 1.99;

Here we create a variable called cost and place the value 1.99 inside it. We are then free to use that variable elsewhere in our sketch as well as perform arithmetical operators on it. So for example, the following will add a small value to the contents of cost:

cost = cost + 0.20; // Adds 20 pence to cost.

Valid arithmetical operators include +, -, \* and / for addition, subtraction, multiplication and division respectively.

For an example of how we might include some variables in a Processing sketch, consider the following, which draws a circle whose colour changes according to the mouse position:

// Draws a coloured circle at the current mouse position.

// Jo Wood, 29th January, 2015

void setup()

{

size(400,250);

// Thick grey border around features.

strokeWeight(4);

stroke(80);

}

void draw()

{

// Set the background to light grey

background(230);

// Create 3 variables.

float diameter = width/5;

float xColour = 255\*mouseX/width;

float yColour = 255\*mouseY/height;

// Draw circle that changes colour with mouse position.

fill(128,xColour,yColour);

ellipse(mouseX,mouseY,diameter,diameter);

}

The general structure of this sketch is similar the previous one - it contains two methods: setup() to initialise the appearance of features to be drawn and draw() to do the actual drawing.

In this example, setup() also includes two lines to set the colour of the outline stroke to grey (stroke(80);) and its thickness to 4 pixels (strokeWeight(4);). As usual, the draw() method contains the instructions to do the drawing. The first line sets the background colour to a light grey (230) before any features are drawn in the sketch.

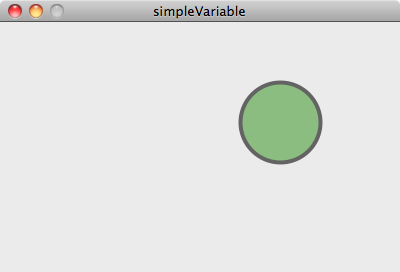
The next three lines create three variables, diameter, xColour and yColour. We can place any numeric value into these variables as well as the results of arithmetical calculations. In the example, we set the variable diameter to be equal to 1/5th of the width of the entire sketch. This gives us a flexible way of ensuring that the circle we are going to draw always maintains the same proportional size relative to the sketch as a whole.

The variables we've chosen to name xColour and yColour are similarly declared, but in this case, they are assigned the value that depends on the mouse position. In particular we scale the mouse X position (left-right) to be between 0 (left) and 255 (right) before storing the scaled result in xColour. We perform a similar scaling with the vertical mouse position inyColour. These two mouse-dependent values are then used to create the green and blue components of the fill()colour in the following line.

Copy the coloured circle code into Processing and running it yourself.   
  
Now try editing the size() parameters so that the sketch is larger (for example, 800 by 500 pixels) and re-running it. See how the circle remains in proportion to the size of the sketch. You can confirm this by changing to a much smaller size (for example, 80 by 50 pixels).

We will see many further examples of creating and manipulating variables in future weeks, but at this stage you should be able to create simple float variables, put numeric values inside them and use those variables to customise the appearance of drawn features.

But for now, we will leave the detail of issuing instructions to build graphical applications and move to look at the bigger picture.

[](http://staff.city.ac.uk/~jwo/datavis/session01/simpleVariable/javascript/index.html)

Moving coloured circle sketch (click to activate).

**1.4 Data Visualization Models**

You've seen some examples of data visualization on the web (e.g. [the Information is Beautiful showcase](http://www.informationisbeautifulawards.com/showcase?acategory=data-visualization&action=index&award=2015&controller=showcase&page=1&pcategory=long-list&type=awards)) and you've been presented with Processing as a coding environment capable of creating interactive graphics. In essence, the aim of this module is to bring together those two approaches to constructing data visualization. At one end is the collection of design principles that determine why and what we choose to visualize. On the other is the set of knowledge and skills necessary to actually create a working visualization.

To help bridge that gap we will be considering some frameworks and models that help structure your thoughts and activities. Two of the most helpful are those created by Tamara Munzner — one of the world's leading information visualization academics, and the workflow model suggested by Ben Fry — the co-designer and lead author of Processing.

**1.4.1 Munzner's Visualization Analysis and Design Framework**

Munzner's framework, that forms the structure of her book *Visualization Analysis and Design*, is centred on three questions that drive data visualization design and production.

**What** data are you wishing to show?  
**Why** is the task, for which visualization contributes, being performed?  
**How** is the visualization constructed?

We will explore all three questions during this module, but it is worth emphasising now that the purpose of the framework is to make the task of choosing from the huge space of data, tasks and idioms (techniques) easer. It recognises that good visualization is as much about thinking about the task it is designed to address as it is about the selection of approrpiate data and the creating of appropriate techniques or *idioms*.

Chapter 1 of her book elaborates on this approach and is recommended reading for all.

**1.4.2 Fry's Data Visualization Workflow Model**

If we want to produce useful data visualizations it is helpful to *plan* the design of our graphics. This is to ensure that the visualization has a useful purpose and its design meets that purpose. As you may have seen when looking at some of the data visualization examples in section 1.2, it is possible to produce graphics that appear, at first view, to be impressive, but actually serve very little purpose.

In Ben Fry's book *Visualizing Data*, he suggests that the process of understanding data begins with a set of numbers and a question. In other words we have some data and we have some task or question we have about the data. For example, we might have some election data and wish to know how geography influences the pattern of voting behaviour, or we might have some software performance statistics and we wish to identify which parts of the software are causing performance problems.

Fry goes on to suggest a set of 7 steps that can help design data visualization to find the answer (Fry, p.5):

1. **Acquire**: Obtain the data, whether from a local file or via a network.
2. **Parse**: Provide some structure for the data's meaning and arrange it into meaningful segments.
3. **Filter**: Remove all but the data of interest.
4. **Mine**: Apply methods from statistics or data mining as a way to discern patterns or structure in the data.
5. **Represent**: Choose the basic visual model of representation, such as a bar graph, list or tree.
6. **Refine**: Improve the basic representation to make it clearer and more visually engaging.
7. **Interact**: Add methods for manipulating the data or controlling what features are visible.

So far we have concentrated on the *represent* and *refine* stages, with some *interaction*. As you progress through this module we will examine all 7 stages of the workflow to see how each stage can help us relate our data to the question or task we have of it. Processing is designed (by Ben Fry) to make all stages of that workflow as easy as possible while maintaining the flexibility to build a wide range of data visualization applications.

**1.5 Conclusions.**

Data visualization is a new and exciting field in modern data analysis and communication. We are currently in a unique position in that there are many vast digital datasets available to us, but that we are generally poorly equipped to handle such data and extract useful information from it. Data visualization offers an engaging and powerful means to explore the complexities of large datasts and to communicate findings and stories about those data to others.

By taking this module, you will become equipped not only to evaluate and design effective data visualization, but also to build your own data visualization applications. We have seen in this first session how to get the software *Processing* to create simple interactive graphical applications. We will continue to use Processing to build increasingly sophisticated applications to answer questions about data. This will be aided by considering Munzner's visualization framework and Fry's 7-stage workflow for data visualization construction.

**Recommended Reading**

*If you are unfamiliar with writing computer programs, you may find Reas and Fry (2010) a good simple introduction. In particular for this session*:

**Reas, C. and Fry, B.** (2010) Chapter 4: Variables, pp.37-50 in *Getting Started with Processing*, O'Reilly

*For an introduction to Munzner's visualization framework:*

**Munzner, T.** (2015) Chapter 1: What's Vis and Why Do It?, pp.1-19 in *Visualization Analysis and Design*, CRC Press

*For an introduction to data visualization with Processing and Fry's 7-stage workflow model*:

**Fry, B.** (2008) Chapter 1: The seven stages of visualizing data, pp.1-18 in *Visualizing Data*, O'Reilly

**Fry, B.** (2008) Chapter 2: Getting started with Processing, pp.19-30 in *Visualizing Data*, O'Reilly

Last modified: Tuesday, 26 January 2016, 9:01 PM