5. Effective data visualisation

Kimbal Marriott

5. Effective data visualisation

Kimbal Marriott

Contents

1
Effective data visualisation: Overview

This is the fifth module in the Data Exploration and Visualisation unit. In this module you will learn about the human visual system and how humans communicate. By understanding these you will be able to design more effective visualisations that take into account human perceptual and cognitive strengths and limitations.

Aims of this module

After completing this module you will:

- understand how the human visual system works and how this impacts on the design of effective visualisations;
- understand data visualisation as a communication act and the implications of this for effective visualisation design;
- appreciate the need for careful design when visually communicating the results of data analysis to stakeholders;
- be able to design effective data visualisations for communicating the results of data analysis;
- be able to use D3 to construct interactive visualisations.

How to study for this module

In this module we draw on books, journal and conference articles as well as material in the public domain, including quite a few videos.

In this module there is one assessment activity:

- Programming a simple interactive visualisation using D3.

2
The human visual system

About half (mainly the rear half) of the human brain is devoted to processing visual information. Within that, the lower (ventral) section is primarily concerned with the analysis *what* (visual features - colour, size, shape...) and the upper (dorsal) section is mainly concerned with *where* (spatial location, relative positions of objects). "...visual information and spatial information appear to be processed differently and separately from each other" (Knauff, 2013 (https://mitpress.mit.edu/books/space-reason)).

The visual system is the product of millions of years of evolution from simple light sensitive cells to the complexity of the human eye. Clearly human vision did not evolve for data visualisation as this is a relatively recent practice. In order to understand the visual system's strengths and weaknesses for data visualisation we need to understand the purposes for which it evolved and how it works.

Overview of visual system

In humans, vision is the primary sense for perceiving the external environment. It is used for navigation, recognising friends, locating food and identifying danger, such as a crouching tiger. The human visual system has a computationally impossible job to do: from a 2D projection on the back of each eye it must reconstruct the shape and position of objects in the 3D world. Computer vision systems are still floundering on this task. Even worse the visual system needs to work quickly: when it comes to crouching tigers a few milliseconds can make all of the difference. As a result the human vision system has inbuilt biases and heuristics for recognising objects quickly: optical illusions reveal how these biases and heuristics can be tricked into making wrong deductions. Effective data visualisation takes advantage of these heuristics to allow the human visual system to quickly perceive patterns and groups.

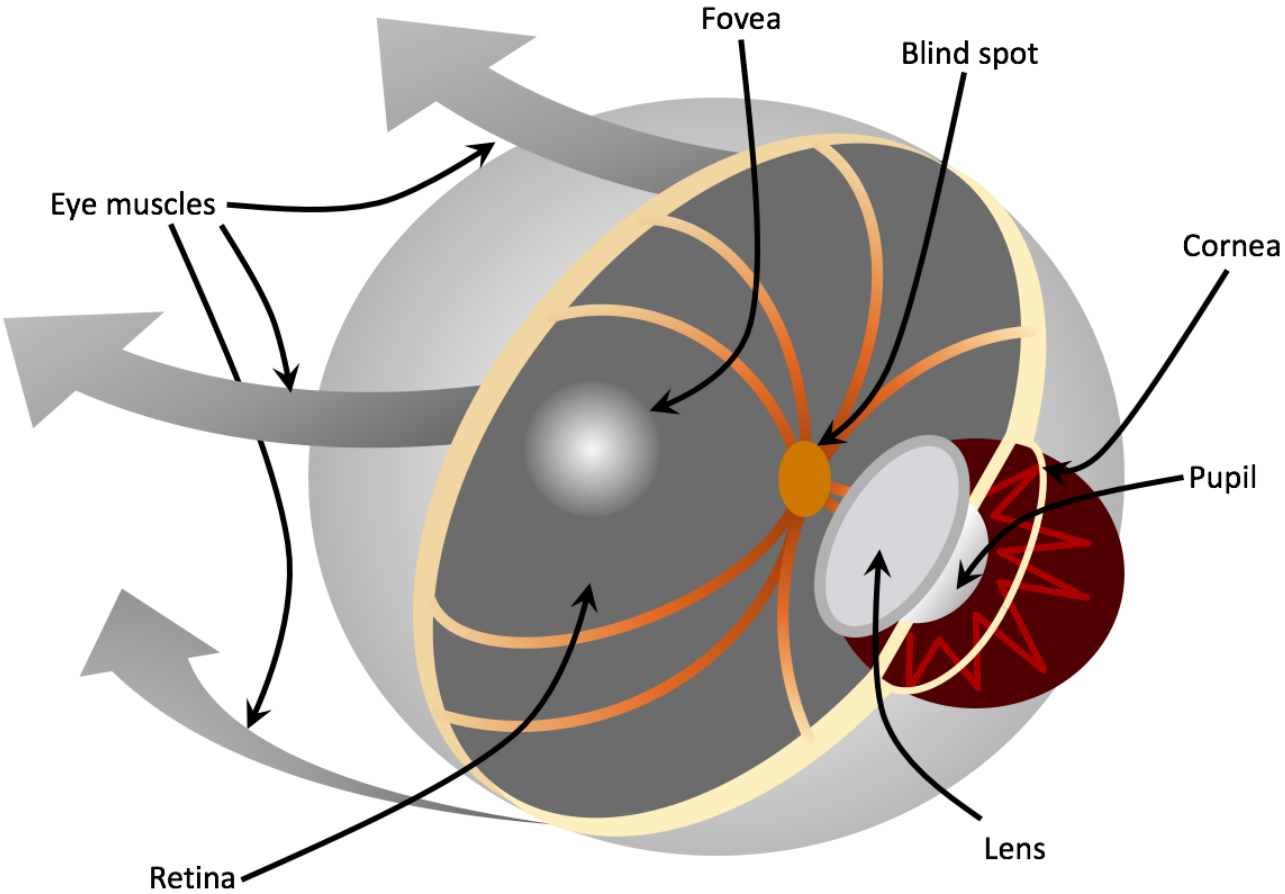
(https://www.alexandriarepository.org/wp-content/uploads/20150929033837/1024px-Tiger-in-kanha.jpg)

A dangerous hidden tiger

The human visual system has 3 main level or stages:

1. Parallel processing to extract low level properties: colour, texture, lines and movement
2. Rapid serial processing divides the visual field into regions of similar colour or texture and achieves "proto-object" recognition of surfaces, boundaries and relative depth. This is driven both top-down by visual attention and bottom-up by low level properties.
3. Visual working memory: object recognition & attention, this is under conscious control

The eye



Main components of the human eye (Based on Figure 2.10 from Information Visualization - Perception for Design by Colin Ware, 2013)

Light enters the human eye through the pupil and then passes through the lens which focuses the (inverted) image onto the retina at the back of the eye. The retina contains two kinds of light sensitive cells: *rods* and *cones*. There are about 100-120 million rods and 6 million cones. Rods are very sensitive to light but only see in monochrome and are not very acute. Cones are less sensitive so do not work well at night but see colour and are more acute. Cones are primarily responsible for day-time vision.

Cones are not distributed uniformly across the retina. Most of the cones are in a small area called the *fovea* which is responsible for detailed vision.

- We see the image falling on the fovea clearly. This corresponds to about 2^{o} of vision which is about an area 2cm by 2cm at arms length
- The rest of the eye provides peripheral vision

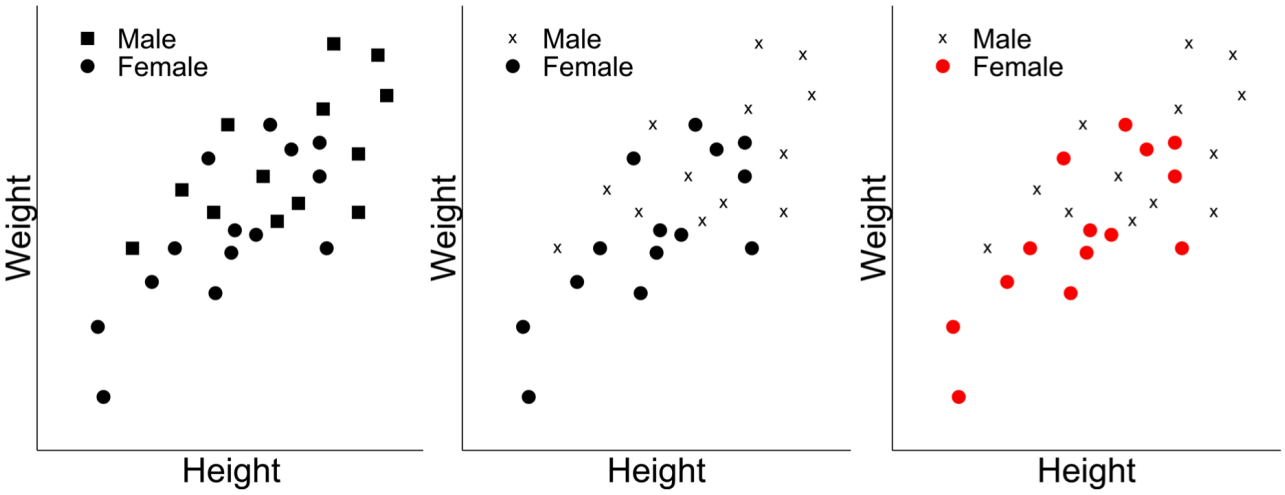
While we believe that we simultaneously see all regions of a data visualisation in detail this is not really true: our eye rapidly darts around the image, fixating on a different region for a few milliseconds and then moving on. The visual system stitches these detailed images together to create an illusion that we see the whole graphic in detail. Nonetheless peripheral vision allows us to see a much larger region in coarse resolution and can direct attention to changes and movement.

Marks and channels

Graphics are made up of *marks*, the basic graphical elements such as a glyphs, lines and regions. A mark's visual appearance and spatial attributes such as position, shape and size are given by *visual variables.* Information graphics map data attributes to these visual variables. Low-level visual processing uses different neural pathways to process different visual variables. These pathways are often called *visual channels*. Different pathways are used to detect motion, orientation, texture, colour and size. This means that these channels are perceptually distinct.

Where possible different channels should be used to encode different attributes, rather than using the same channel such as colour to encode multiple attributes. It does not hurt to use redundant encoding.

Lines and shapes are recognised by specialised cells called Gabor receptors. Different receptors respond to different frequency and orientation of input lines. This means that symbols should be as distinct as possible from their background and from one another in terms of their components spatial frequency and orientation.

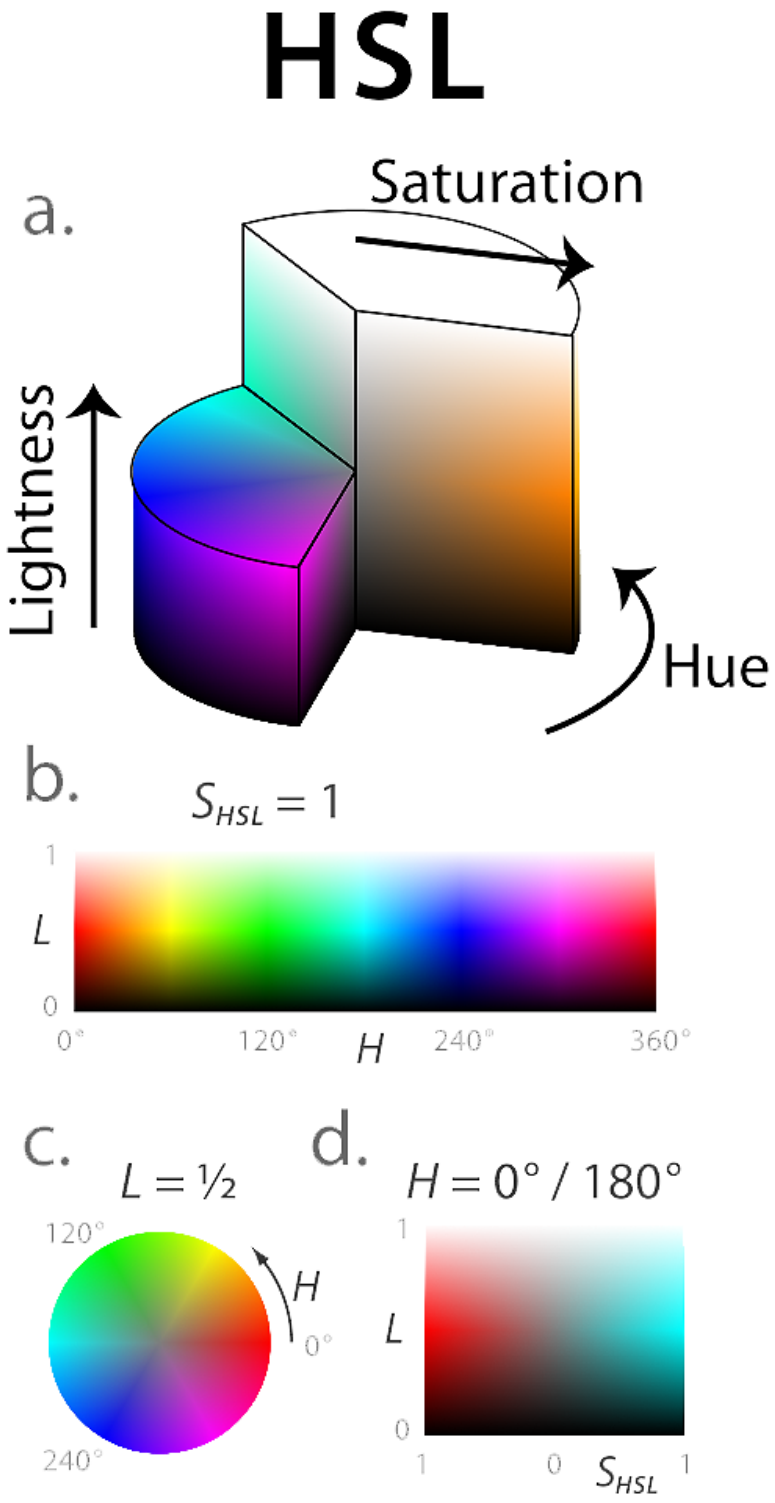


Feature channels can be used to make symbols more distinct from one another. Crosses are perceptually distinct to filled circles, so it is easier to separate males from females in the graph in the middle than the graph on the left. The graph on the right use redundant color coding in addition to more distinctive shapes making it even easier to distinguish the two sexes. (Based on Figure 5.8 from Information Visualization - Perception for Design by Colin Ware, 2013)

Colour

Colour is actually composed of three different channels. Cones provide colour vision: they come in three

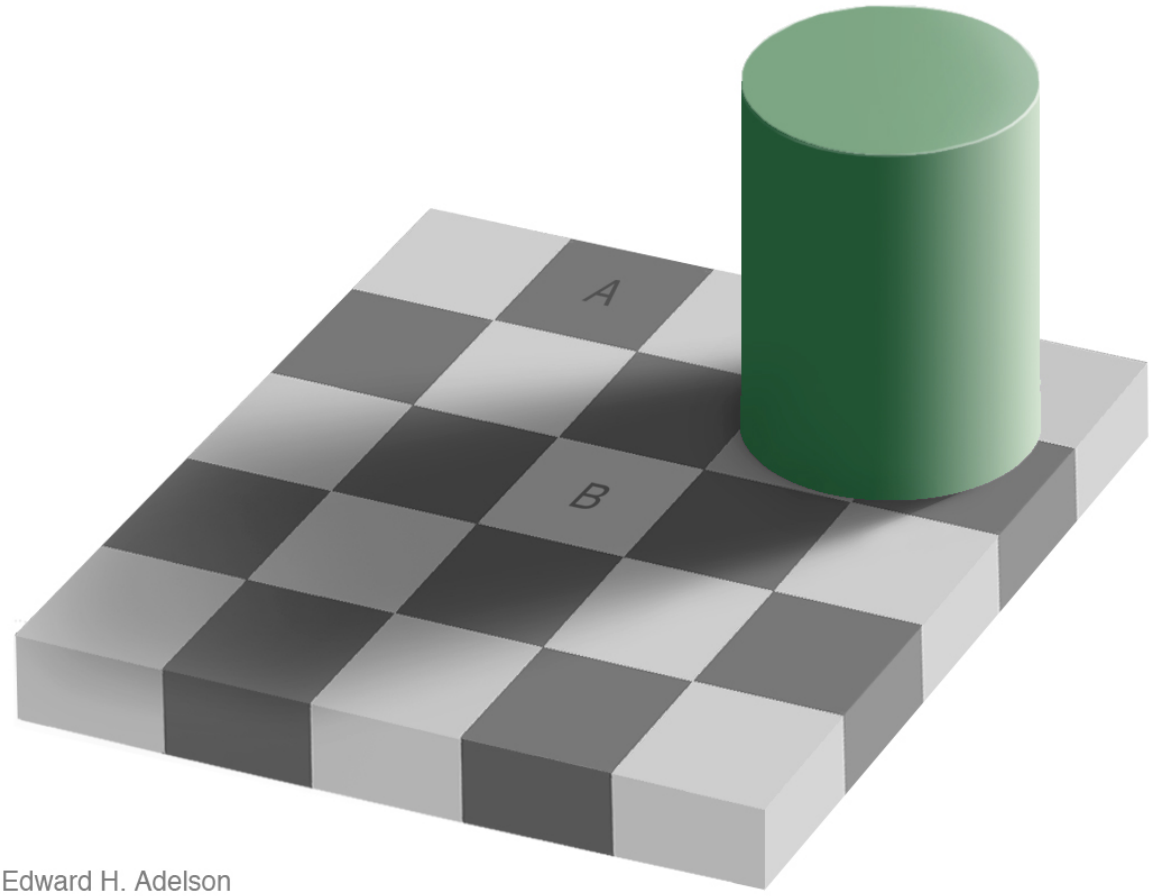
varieties each with a peak response to a different light frequency within the visible light spectrum. Low-level visual processing encodes these in terms of three opponent colour channels: red to green; blue to yellow and, the most important channel, black to white which encodes *luminance*.



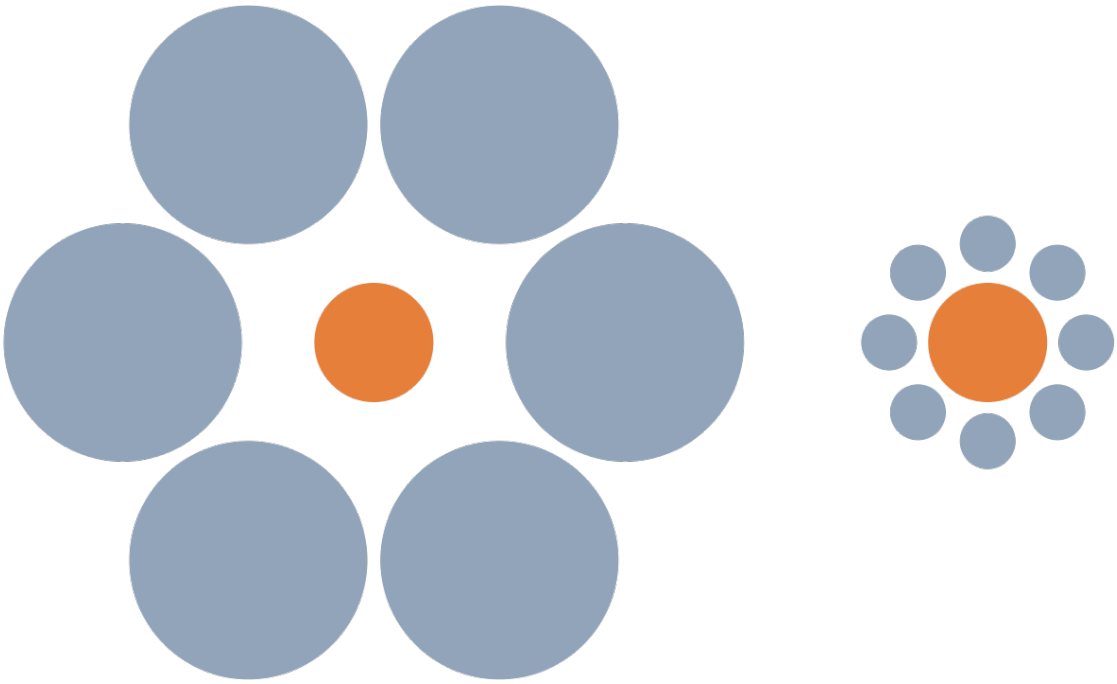
(https://www.alexandriarepository.org/wp-content/uploads/20150929033837/hsl.1.png)
Top: cut-away 3D models of HSL (a); bottom: two-dimensional plots showing two of a model's three parameters at once, holding the other constant (b, c, d). (Based on figure 1 on HSL and HSV page of Wikipedia)

In data visualisation it is common to think about colour in terms of the HSL colour space: H for hue-the choice of pure colour, S for saturation-the amount of white mixed with the colour, and L for lightness-the amount of black mixed with the colour. Another colour space that is common in computer graphics is the RGB system which codes colours in terms of the amount of red, green and blue. While HSL is not ideal it is a closer match to the actual perceptual system and should be used instead of RGB.

One thing to be aware of is that colours inhibit adjacent colours. This means that the same colour can appear quite different in different contexts. Boundaries between colours help this. You also need to be aware that the amount of area affects perception. Use low-saturation lighter colours for large background regions, higher-saturation darker colours for small foreground shapes or regions.

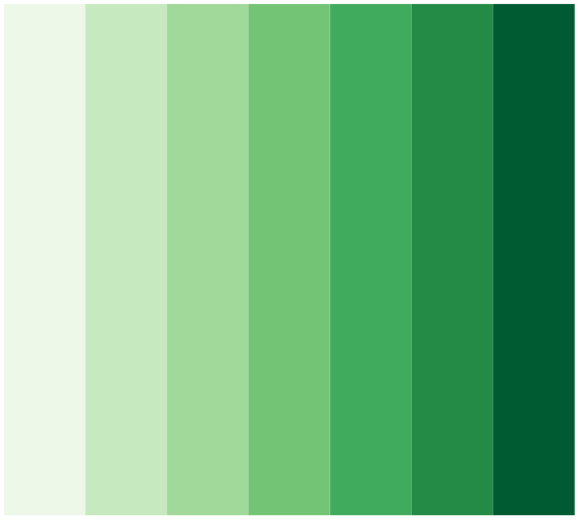


(https://www.alexandriarepository.org/wp-content/uploads/20150929033837/checkershadow_illusion4full.jpg)
Colour inhibition: the colors of the squares labeled A and B are the same! If you don't believe this download the image and check the RGBs value in a photo editor. (This figure is from Edward H. Adelson)



(https://www.alexandriarepository.org/wp-content/uploads/20150929033837/1280px-Mond-vergleich.svg.png)
There is also size inhibition. Believe it or not the two orange circles in the center are in the same size. (figure is from Wikipedia)

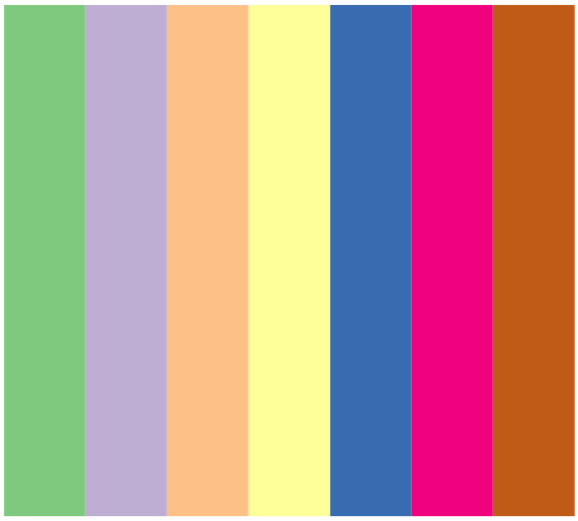
Colour choice is quite difficult. Luminance and saturation are automatically interpreted as ordered while hue is not. However hues that vary along only the red-green or blue-yellow channel do have a natural ordering. Fortunately many colour maps or palettes and tools have been developed to help in data visualisation design. One of the most commonly used is by Cindy Brewer**.** Her ColorBrewer (http://www.colorbrewer.org) tool enables selection of handcrafted color schemes for various tasks. Her colour schemes are also available for use in R (RColorBrewer, examples below). Her tool distinguishes between three different kinds of data: sequential (ordered but ascending from a single least value), diverging (ordered but ascending and descending around a neutral value), and qualitative (categorical). Ware (2013) also presents a number of colour maps.



Greens (sequential)

(https://www.alexandriarepository.org/wp-content/uploads/20150929033837/sequential.png)

BrBG (divergent)
(https://www.alexandriarepository.org/wp-content/uploads/20150929033837/diverging.png)



Accent (qualitative)
(https://www.alexandriarepository.org/wp-content/uploads/20150929033837/qualitative.png)

Colour-blindness & accessibility

One thing to aware of when using colour is that colour blindness is quite common. About 10% of males and 1% of women have some kind of colour blindness. The different colour channels explain the different kinds of colour blindness. Most commonly differentiation on the red-green channel is reduced (about 8% of men but much less common in women) while blue-yellow channel differentiation is much less common and not sex related.

When designing colour schemes the easiest strategy is to ensure that hue is not the only channel used to encode information. For categorical data choose colour maps that vary in luminance or saturation as well and if possible avoid colour maps that emphasise red-green. Sites such as http://www.color-blindness.com provide on-line tools to show what an image looks like with different kinds of colour blindness.

Maureen Stone an expert in the use of colour (who now works for Tableau) introduced the slogan *Get it*

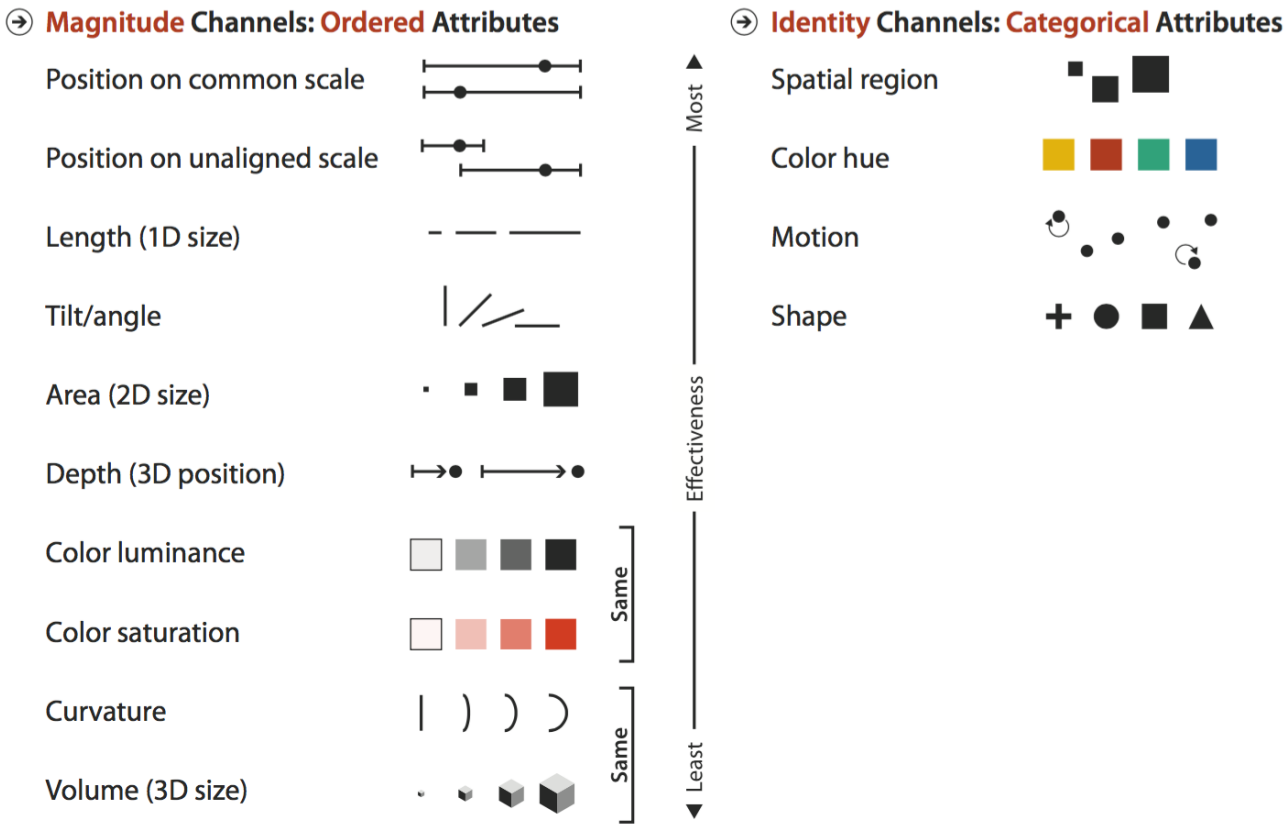
Right in Black and White. She suggests that you should develop visualisations in black and white first, ensuring that the important aspects of the visualisation are still legible when the image is rendered in greyscale. Hue and saturation, i.e. colour, is added later, to provide redundant or secondary information.

Which visual variable should I use?

Visual variables are not interchangeable: the same data attribute encoded using different visual variables will not be perceived as effectively. Different variables vary in terms of

- *salience* - how quickly they are noticed. For instance movement is more salient than orientation.
- *discriminability* - how many distinct values can you encode without confusion to the user
- *accuracy* - how easily can you compare different values.

Experiments have shown that the visual variables commonly used for encoding quantitative or categorical data vary greatly in accuracy and discriminability . The following figure summarises their effectiveness

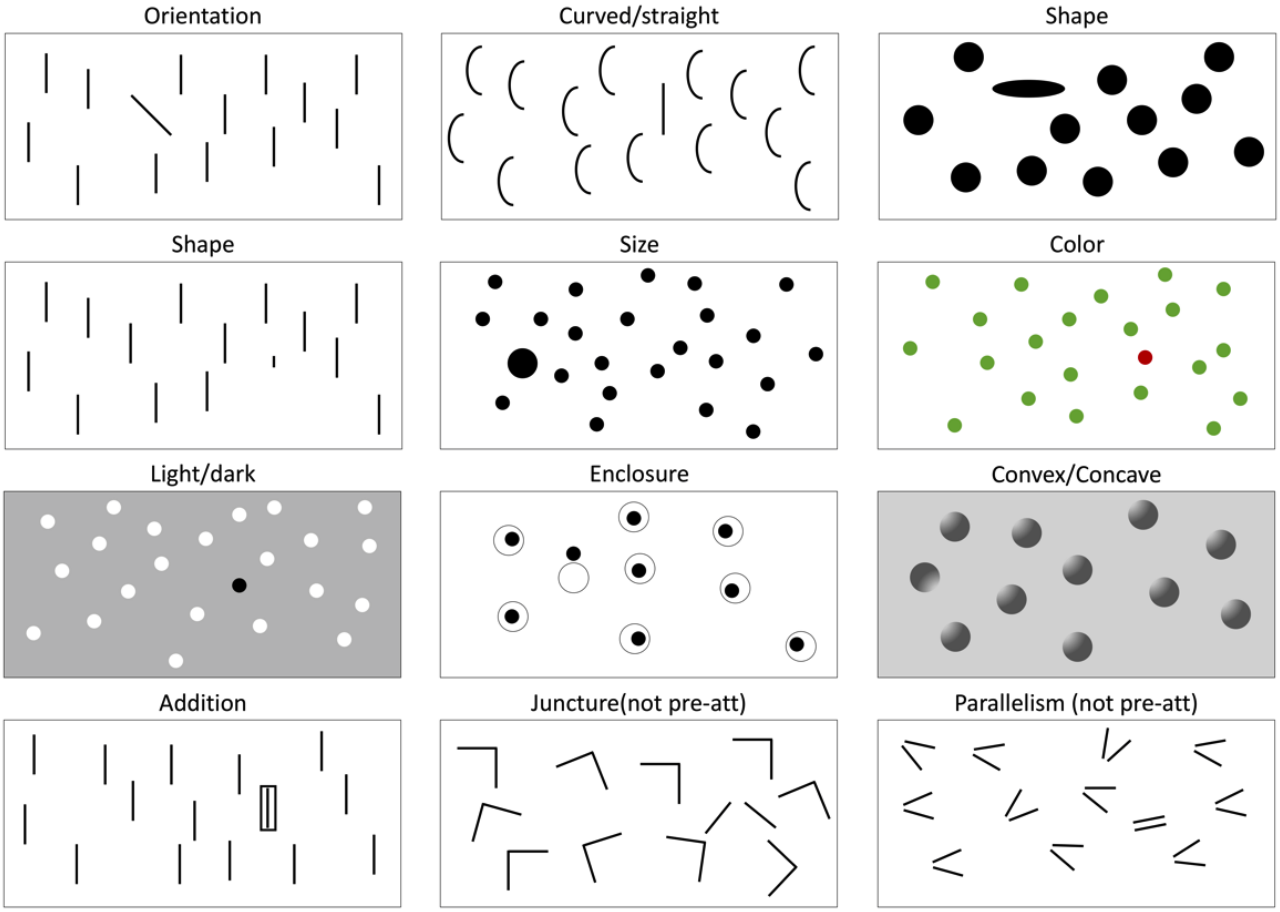


Effectiveness of different visual channels. (Fig 5.1 from Visualization Analysis and Design by Tamara Munzner, 2014)

What this means is that you should use bar charts rather than pie charts or doughnut charts and that if occlusion is not a problem then a prism map is more effective than a choropleth map for ungrouped data. And you should never, never use a 3D pie-chart!

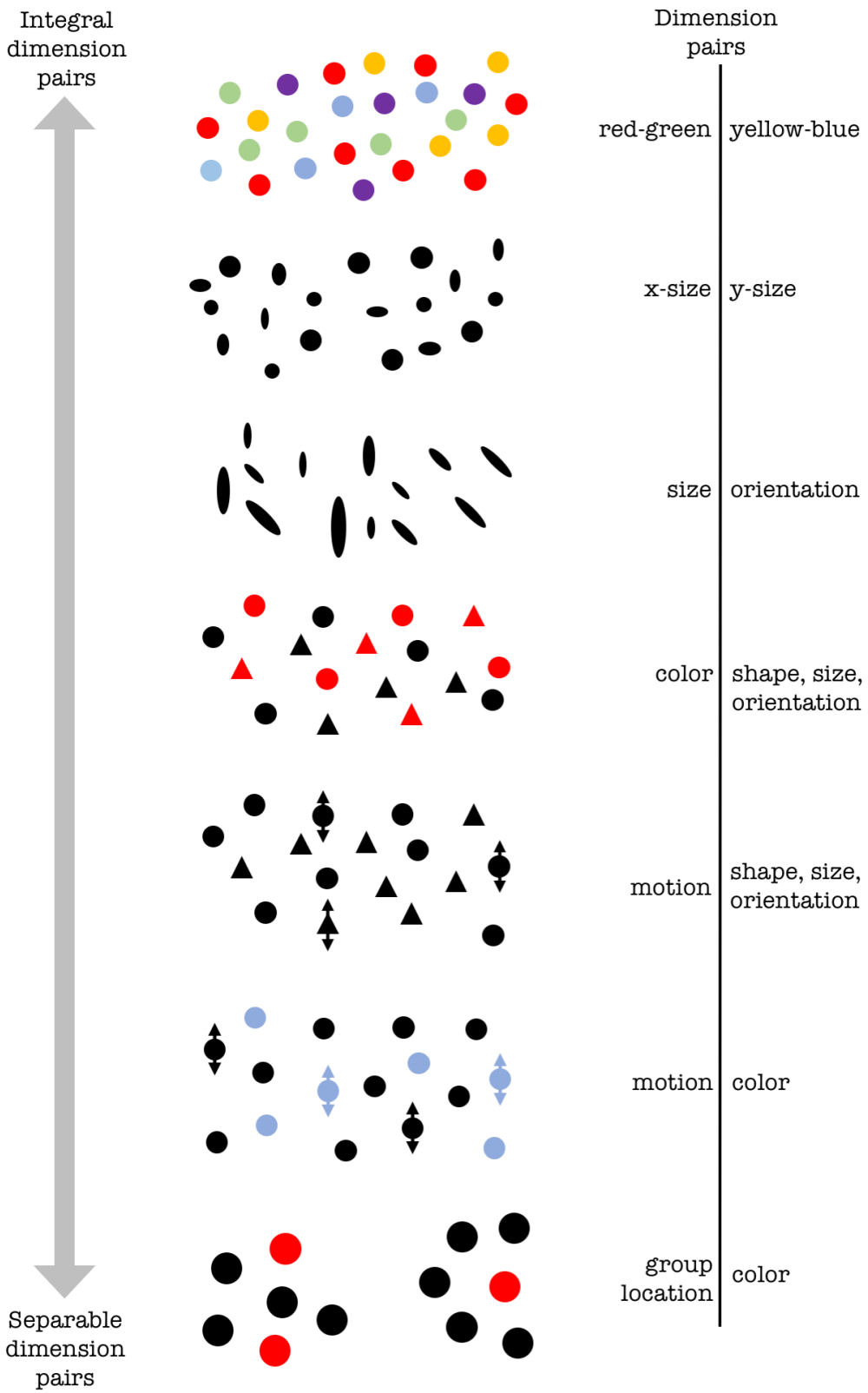
Related to discriminability is the degree of "visual popout"- that is how well target items standout from the other items. Visual pre-attentive processing occurs in parallel and objects that are pre-attentively distinct from the other items are are found quickly and the time taken is independent of the number of non-target items. Without visual popout target items must be found using a conscious serial search through all items and so the time taken to find target items depends upon the number of non-target

items. Many channels support visual popout, at least to some extent. They include line orientation, length and width, size, curvature, spatial grouping, blur, annotation, color, motion and position. Pre-attentive cues should be used when directing or attracting attention or to show search results.



Examples of glyphs showing different kinds of visual pop-out. (Based on Figure 5.11 from Information Visualization - Perception for Design by Colin Ware, 2013)

You cannot choose visual variables independently of one another as the underlying visual channels interfere with other to varying degrees. At one extreme are visual variables like colour and location that are *separable* in the sense that they have very little interference and at the other are variables like the red-green and yellow-blue colour channels that are *integral* and have high interference. All things being equal you should use visual channels that interfere as little as possible.



(https://www.alexandriarepository.org/wp-content/uploads/20150929033837/ware.fig_.5.23.coding.2.png)

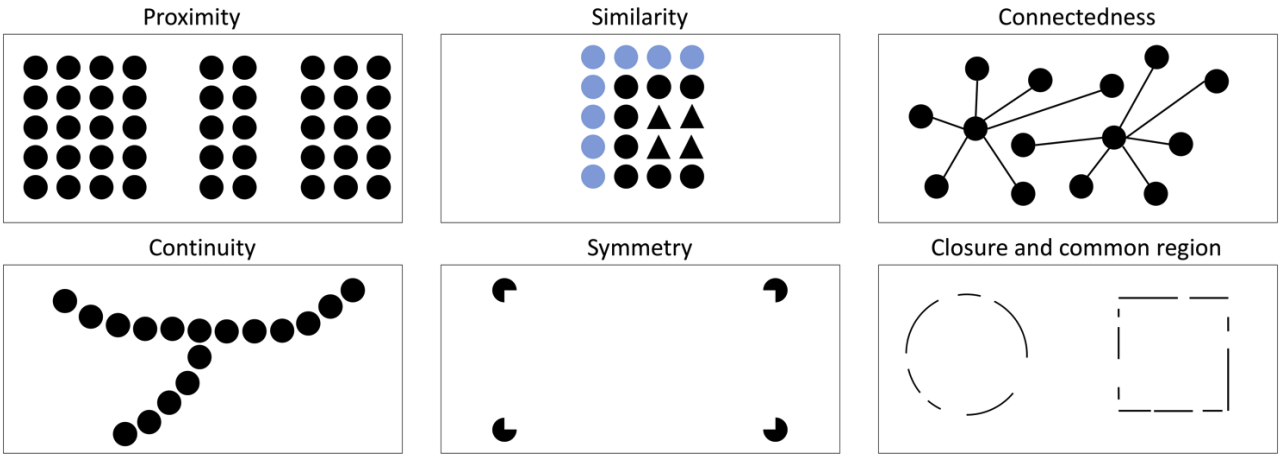
Examples of glyphs coded according to two display attributes. At the top are more integral coding pairs. At the bottom are more separable coding pairs. (Based on Figure 5.23 from Information Visualization - Perception for Design by Colin Ware, 2013)

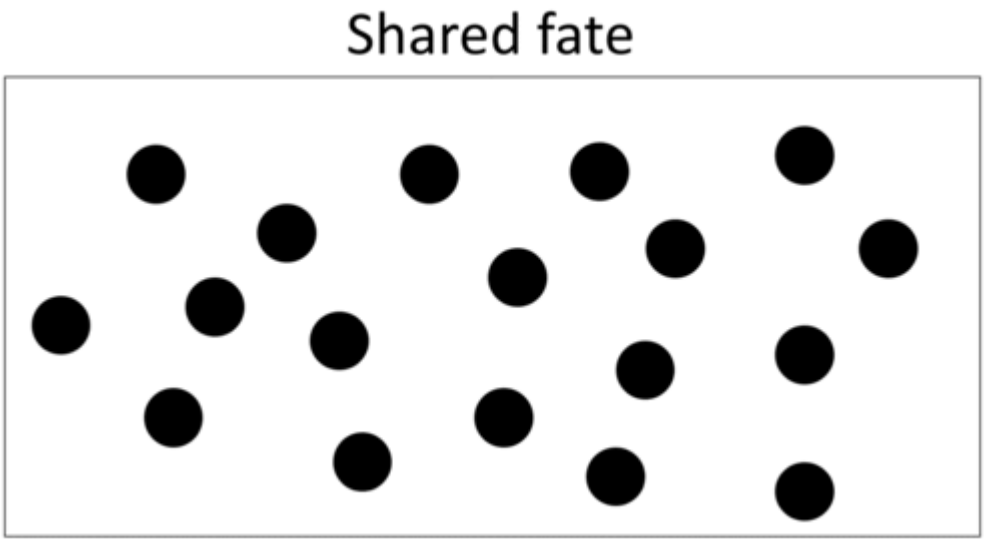
Grouping

Colour, line orientation and frequency, stereoscopic depth and motion are identified in the first stage of visual processing. In the next stage contours, regions and foreground and background are identified. This is the stage in which pattern perception is used to extract objects from low-level visual features.

Perception of visual patterns was first seriously studied in the early 20th century by a group of German psychologists who identified a set of laws of pattern perception which were called the *Gestalt laws* since Gestalt is the German word for pattern. Based on this research we now know there are many ways in which people automatically organise elements

- *Proximity*: Elements that are close together form groups.
- *Similarity*: Elements that are similar in some way such as colour or shape form a group
- *Connectedness*: Connection by lines is a powerful way of grouping elements
- *Continuity*: We tend to group regions and lines to form smooth and continuous shapes.
- *Symmetry*: We are good at recognising bilateral symmetry, especially around a horizontal or vertical axis and group the symmetric lines together to form an object.
- *Closure and common region*: we like to see closed contours and will mentally extend lines to close them. Being "inside" a closed contour is a very powerful grouping principle
- *Shared fate*: Elements that move together are grouped together.





Examples of different Gestalt Principles for grouping elements.

These principles capture the heuristic rules that the human visual system uses to group the lines and

regions of similar colour and texture in 2D in order to segment the image into foreground and background and into different objects so that they can understand what they are really looking at in the 3D world.

Information graphics take advantage of these heuristic rules to help us see patterns etc. For example, scatter plots make use of proximity, node link diagrams make use of connectedness, Venn diagrams of common region and paired bar charts make use of symmetry.

Perceiving 3D

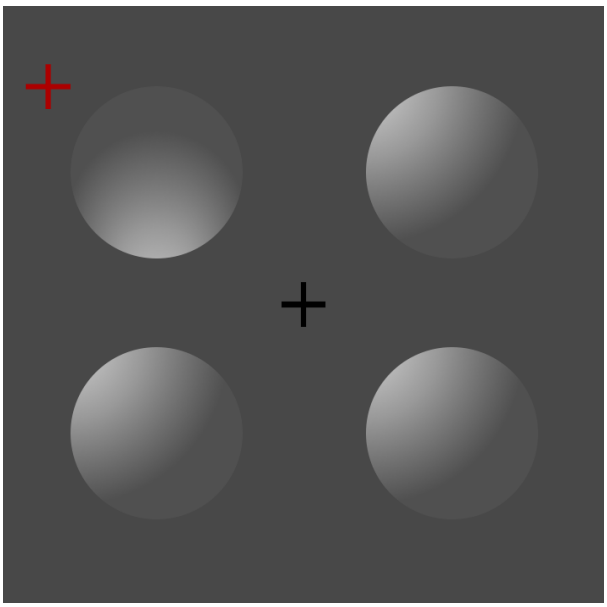
An important part of understanding our 3D environment is the way in which the visual system extracts information about depth from what are essentially two 2D visual images, one for each eye. A wide number of different *depth cues* are used, most of which are now used to create lifelike immersive 3D visualisations.

(https://www.alexandriarepository.org/wp-content/uploads/20150929033837/angkor.png)

Scene from 3D animation constructed by Monash academic Tom Chandler showing how the Cambodian temple complex of Angkor may have looked.

Monocular static cues

- *Occlusion*: this is the most important depth cue-objects in front obscure those behind.
- *Linear perspective*: foreshortening, parallel lines converging to a point. We see the sides of the road converge and that people get smaller in the distance.
- *Shape-from-shading*: We see this illustrated below - here light from above is suggested by the shading. Concave or convex dimple shapes are suggested by shading. In the Angkor image above it used to show the shape of the elephant's head.



- *Shape-from-texture distortion*: Wire frames make use of this to show shape
- *Cast shadows*: Cast shadows give a clue about height above the object on which the shadow is cast.
- *Familiar size*: Familiar objects allow us to judge distance because we know how big they actually are.
- *Depth of focus*: our eyes change focus to bring the image of the object we are looking at into sharp focus on the fovea. Objects that are closer or further away are blurred giving an ambiguous clue as to their depth.

Monocular dynamic (moving picture)

- *Structure from motion*: rotation and movement of an object relative to the observer allowing them to see it from different points of view is an extremely important depth cue.

Binocular

- *Vergence angle*: When the eyes look object at an object at a certain depth the visual system can make use of the difference in angle between the line-of-sight vectors of the two eyes to measure depth of objects that are close by (roughly within arm's length)
- *Stereoscopic depth*: The visual system can make use of small differences between the images on each eye to see depth. 3D TVs and displays provide stereoscopic vision. While stereoscopic vision (in combination with the other cues) can provide a sense of truly immersive 3D it is only one of many depth cues and is actually not that important. Something like 20% of people do not have stereoscopic vision and many never notice its absence.

Not all of these depth cues are needed to create realistic 3D graphics and may not be needed at all in some tasks. Ware (2013) provides a more detailed analysis. Occlusion is the most important depth cue. I think structure-from-motion is the next most important and can also mitigate the disadvantage of occlusion hiding information. Ware recommends that if structure-from-motion is used then so should occlusion, linear perspective and texture-distortion or else it looks strange.

Data visualisations also make use of more artificial depth cues to show depth. These include showing gridded ground and side planes to show perspective distortion. An extra cue is to projecting the 3D data onto these planes. In the case of 3D scatter plots it is common to drop lines to the ground plane so that the points look like pins.

An important question is when to 2D or not 2D? The general rule is that you should use as few dimensions

as is required. Thus if if you are simply comparing the magnitude of a single attribute use only a single dimension and plot the values on a uniform scale. There is no need for 2D in this case.

In the case of 3D it should be used when visualising inherently three-dimensional structures such as buildings and other physical objects and flows. This is why immersive 3D is so important in scientific visualisation.

The use of 3D for abstract data visualisation is less easy to justify and by default you should use 2D. The disadvantage of 3D is that occlusion hides information and the perspective distorts size, making it difficult to compare magnitudes. Interaction is also more difficult. For this reason 3D bar charts are a very bad idea. However my view is that 3D will be used more frequently in abstract data visualisation when low-cost 3D visualisation technologies, such as the HTC Vive, Oculus Rift or zSpace, that allow the user to naturally vary their viewpoint become available. By allowing the observer to move relative to the graphic the problems of occlusion and perspective distortion are mitigated. They will be useful when looking at actual 3D visualisations like 3D scatter plots, prism maps, space-time cubes and congruent 2D surfaces drawn in 3D (sometimes called 2 1/2 D).

Visual attention and working memory

In the third and highest level of visual processing, visual objects are held in working memory while the viewer performs some task such as finding the shortest route between two cities. At this level processing is conscious and sequential. Only a few objects are held in memory at one time.

We use several types of memory in visual processing: *Iconic memory* (aka visual cache) which is essentially a very short-term snapshot of the image on the retina; *visual short-term memory (STM)* which holds the visual features of objects of immediate attention; *spatial STM* which holds the position/location of the objects*;* and *long-term memory (LTM)* which holds memories retained from previous experiences. There are similar kinds of memory for other modalities such as echoic and verbal working memory for sound. (e.g. Baddeley, 2007 (https://books.google.com.au/books/about/Working_Memory_Thought_and_Action.html?id=P2UQAQAAIAAJ&redir_esc=y)).

Visual working memory holds visual objects from long-term memory as well as those on the screen. Actually visual working memory is probably not distinct from long-term memory, it is simply the current activated long-term memories. We can also think of the visualisation on the screen as a different kind of memory: *external visual memory*.

One of the most surprising finds of psychologists has been how few objects can be held in our working memory, somewhere between 3 and 5, depending upon task. And we only remember 3-5 objects if we are concentrating, usually only 1 or 2 objects are remembered.

To most people this limited capacity seems extraordinary, as we feel as if we have a rich internal representation of the world we are seeing. This is however not true. *Inattentional change blindness* is a powerful demonstration of our lack of memory capacity. Because we remember so little, we do not notice large changes between what we see in one view and the next. Change blindness is graphically shown in this video (http://www.youtube.com/watch?v=VkrrVozZR2c). The experiment is about 1:40 into the video but I encourage you to watch the entire video.

The limited capacity of working memory has strong implications for visualisation. In particular it means that we are better to encode multiple attributes into one visual object rather than using separate visual objects for each attribute, since, if multiple data is integrated into a single object, more information can

be held in visual working memory. For instance if we are examining wind direction, temperature and wind strength we are better off encoding this in a single glyph such as an arrow whose orientation gives the direction, colour the temperature and length or width the strength rather using three different glyphs.

Visual attention is the key to understanding how information flows between the different visual processing stages. As the viewer performs the task their attention turns to different parts of the image. When they move their eye to focus on a new region, subconscious parallel processing of stage 1 extracts low-level properties. Visual attention guides stage 2 processing to extract the surfaces and features of the objects that the viewer is now looking at and stage 3 processing recognises these objects and places those being attended to in working memory. However, visual attention may also be driven by stage 1: if a light blinks in peripheral vision, this will be noticed subconsciously ad the viewers attention will be drawn to it.

Controlling attention is a key-part of effective visualisations. You need to direct attention to the salient parts of the display.

A recent theory suggests that for tasks involving visual reasoning, the spatial aspects of a display are the most important - too much visual detail reduces performance (*visual-impedance hypothesis*, Knauff & Johnson-Laird, 2002 (http://www.kyb.tuebingen.mpg.de/fileadmin/user_upload/files/publications/pdfs/pdf2442.pdf)). This is consistent with calls from some information visualisation designers (e.g. Tufte (http://www.edwardtufte.com/tufte/books_ei)) to produce clean, minimalist displays from which irrelevant detail ('chartjunk') is eliminated.

Summary

In this topic we have investigated how the human visual system works. We have seen how it has 3 main stages: low-level feature extraction; region, depth and boundary recognition; and visual working memory and object recognition.

The design of good visualisations needs to take into account the perceptual and cognitive limitations of the visual system.

- Different visual channels should be used to encode different attributes and the choice of channel is important
- Low-level feature recognition occurs in parallel and supports "visual popout".
- Colour schemes and interfaces should be designed to cater for colour blindness.
- Pattern matching and grouping makes use of visual heuristics
- Use 1D in preference to 2D and 2D in preference to 3D unless there is a strong reason not to.
- We have extremely limited working memory: attention should be directed to salient parts of the display.

FURTHER READING

The material in this topic is mostly based on

Munzner, Tamara. *Visualization Analysis and Design*. CRC Press, 2014.

Ware, Colin. *Information visualization: perception for design (3rd Ed.)*. Elsevier, 2013.

Further reading is

- Chapter 10 of Munzner, 2014.

3
Visual communication

Understanding the human visual system is one part of understanding how to create effective data visualisations. Another component to this is understanding human language and how humans communicate. This aspect is important when designing data visualisations for communicating what you have found to stake holders as they will subconsciously understand the data visualisation as a "communication act." In this topic we look at this aspect of visualisation design.

Effective communication

Most cognitive psychologists and linguists now believe that humans have an inmate instinct to learn and use spoken language. All human languages share a common "deep" structure and children who are deaf and so not exposed to spoken language will, if they are given the opportunity, spontaneously invent sign languages so that they can communicate. Amazingly although sign language is understood visually and produced using hand gestures, the same parts of the brain are used for understanding and producing sign language as are used for spoken language. Humans really want to be able to talk to one another.

When we communicate with each other there are a number of assumptions that are true for spoken language, written language, graphic novels, movies and sign language:

- *Relevance*: The information that is provided is relevant to whatever is being discussed and that neither too much or too little information will be provided.
- *Appropriate knowledge:* We tailor the communication to the listener or reader, taking into account their knowledge and also cultural expectations.
- *Directing and holding attention*: The discussion is sequential, with clear indications as to what is important and when there has been a change in topic.

Paul Grice (http://www.ucl.ac.uk/ls/studypacks/Grice-Logic.pdf), a philosopher of language, proposed several 'conversational maxims':

- the maxim of quality - provide information that is true, do not say that which you know to be false or for which you lack sufficient evidence
- maxim of quantity - provide enough information as is required but not more than is required
- maxim of relevance - ensure that the information you provide is relevant
- maxim of manner - avoid obscurity, ambiguity, be brief and be orderly

These points hold equally true for the visual language of data graphics. They mean that you need to be very clear about what message you are trying to convey when designing a data graphic for communication and who your intended audience is. It means only showing data that is relevant to that message and choosing charts and graphics that the user is familiar with, or at least ones which they can easily understand and providing titles and legends etc that explain what the graphic is about. and how to read it. It means using visual tricks to direct attention to the important part of the graphic. The reader's attention will be drawn to the most visually striking components of the graphic so these should be the most important parts of the message, i.e. highly salient. We have seen from the last topic how to make parts of a display pop-out. Another technique is to use annotations to direct attention. Readers expect changes in appearance from, say interaction, to be relevant and informative.

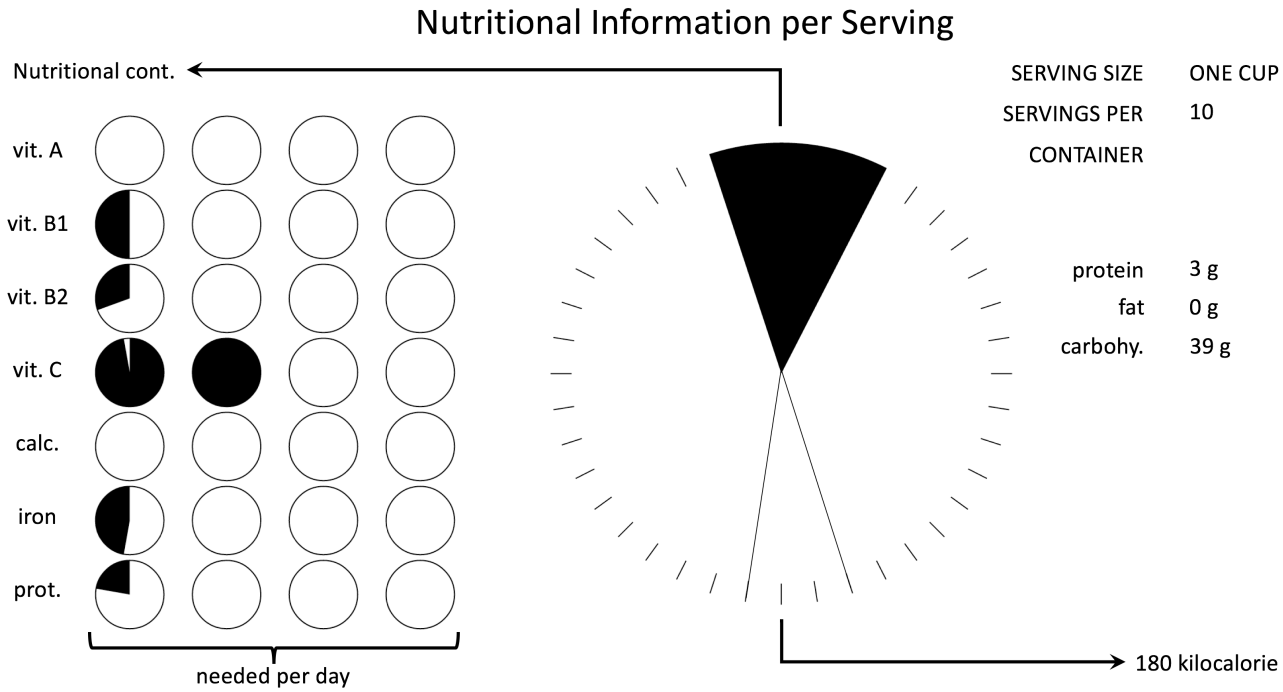
In many situations a mix of graphics and text is required in order to communicate more complex messages. Relevance, appropriate knowledge and narrative need to be considered for the overall mix,

with clear visual signals guiding the reader on which are the most important elements and in what order to read them.

When designing an effective data visualisation it is crucial to take into account the limitations of the human visual system and cognitive system discussed in The human visual system. In particular you should choose visual variables and representations that facilitate

- *Discriminability*: allow the reader to easily discriminate and compare data
- *Perceptual organisation*: encourage the perceptual system to group data in meaningful ways, such as by using Gestalt principles and different channels
- *Compatibility*: The conventions used are compatible with what they represent, e.g. use line graphs for continuous data, bar charts for discrete measurements and "more" of a visual variable such as length means "more" of the underlying quantity being conveyed.
- *Capacity limitations*: take into account limitations in working memory and processing limits.

Kosslyn (2006) presents a graphic that was designed to present nutritional information to consumers on food packaging. Apparently the US seriously considered making the use of this display mandatory. Please take a look at the display and see if you can understand it.



Proposed nutritional display graphic. (Based on Figure 1.8 from Graph Design for the Eye and Mind by Stephen M. Kosslyn, 2006)

He identifies the following problems

1. The reader is likely to think that the small circles are pie charts but in this case it is the group of 4 circles that shows the whole. This violates the assumption that the reader has appropriate knowledge.
2. The circles are arranged so that they are perceptually grouped into 4 columns instead of 7 rows when in fact the rows are the logical way to group the circles.
3. The center panel also violates the assumption of appropriate knowledge as it is a graphic that the user has almost certainly never seen before and probably hopes never to see again. While it looks like a pie chart it is not and actually consists of two separate graphics: a black wedge at the top

giving the nutritional content (vitamins+minerals) and white wedge at the bottom giving the number of calories. Even though the two wedges are aligned and on the same background and so perceptually grouped together that are in fact logically independent. It is impossible to understand the scale being used for each wedge and it is exceedingly difficult to know how many tick marks the black wedge subsumes.

4. It is also very difficult to know how to read the display and which pieces of information are the most important.

In terms of the Grician maxims, the graphic violates those of manner (it is obscure) and quantity (it provides more information than is required).

Narrative visualisations

Online interactive graphics are increasingly used by journalists and data scientists to communicate the patterns and trends they have found in data. Publishing to the web is made easy by tools like Tableau and with D3 it is possible to create powerful and sophisticated visualisations. Such graphics blur the distinction between exploration and communication and data visualisation researchers and graphic designers are still exploring different ways of using interactive graphics for communicating their findings.

Good examples of these new kind of data visualisation are those being created by the New York Times. Take a look at Budget Forecasts, Compared With Reality (http://www.nytimes.com/interactive/2010/02/02/us/politics/20100201-budget-porcupine-graphic.html?_r=0) from 2010. This explores quite a complex topic: budget forecasting and the reasons why it mostly goes wrong. It uses what is essentially a slide show with interactive slides to guide the reader through the story. Linked text and visualisation along with the linear slide show make it clear in what order to read the visual elements and what is important. Interaction allows the reader to explore the data but only in a carefully controlled way.

Segel and Heer (2010) provide an overview of such *narrative visualisations*. They identified seven genres: magazine style, annotated chart, partitioned poster, flow chart, comic strip, slide show, and film/video/animation. These differ in the number of frames and the ordering of the frames. The genres can be combined. For instance, the Budget Forecasts example uses annotated graphs within a slide show format.

They discussed the many different kinds of narrative tactics used in these visualisation, some of which are borrowed from movie making. These visual devices help the reader navigate the visualisation and to appropriately focus attention. They include highlighting to direct attention to salient parts of the display, transition guidance to help the reader move between the different elements in the visualisation without disorientation such as smooth animations, camera motion and visual structuring that helps the reader understand the overall structure and where they are in the narrative.

The final aspect that Segel and Heer (2010) considered is the visualisation structure. This varies in how much the author directs the story and how much freedom the reader has to explore the data to create their own interpretation. The Budget Forecast example is called an *interactive slideshow with single-frame interactivity*. In this structure the overall structure is linear but the reader can explore the data in a single side. They are also free to step backwards and forwards through the narrative. Another well-known example of this kind of visual narrative is Gapminder Human Development, 2005. (http://www.gapminder.org/downloads/human-development-trends-2005/)

Another visualisation structure is the *martini glass*. This begins with questions, observations, or written articles to introduce the visualization but then the narrative expands to allow the reader to freely explore the data (hence the amount of exploration allowed expands like the shape of a martini glass). An

interactive chart with associated text is a common example of this kind of narrative.

The final structure they identified was the *drill-down-story*. This presents a general theme but allows the reader to interactively choose different aspects to the story and drill down into the data. Interactive posters or maps often have this structure. Comparison of Bear Markets, The New York Times, 2008 (http://www.nytimes.com/interactive/2008/10/11/business/20081011_BEAR_MARKETS.html) is an example.

Animation & interaction

It is easy to believe that animation is better than multiple static frames at showing changes. However, it turns out to be more complex. Studies referenced by Ware (2009) suggest that, at least for showing how mechanical devices work, like a flushing toilet, snapshots of the key stages with careful annotations directing attention to changes and direction of movement work more effectively than animations. This is probably because: (1) the snapshots carefully direct and guide the reader's attention, (2) it is easy for the reader to compare different steps and to move backwards and forwards between these, and (3) they encourage the viewer to "mentally animate" components between the different steps which helps deeper understanding,

As a result of these studies, visualisation designers now realise that they need to be more careful when designing animations and changes resulting from interaction. They need to carefully direct attention, use visual continuity to help the reader preserve their mental map of the visualisation, stagger changes so that not everything changes at once and use carefully designed animations to show how the elements move. These tricks are the kinds of visual narrative tactics identified by Segel and Heer.

Summary

The design of effective visualisation for communication is quite difficult and requires understanding the human visual system as well as conventions used in communication. The following summarises the design process:

- The key first step is to to clearly identify what message you wish to communicate and to whom.
- The second step is to decide on the genre, the narrative structure and the presentation technology.
- The third step is to design an appropriate visualisation. Do mock-ups of different designs to explore the design space. Critique the designs taking into account: relevance, appropriate knowledge, directing and holding attention, discriminability, perceptual organisation, compatibility and human capacity limitations.
- The last step is to implement the visualisation and actually check that it is effective by testing it, if possible, with members of the target audience and refining the design

---

FURTHER READING

This topic is based on

Ware, Colin. *Information visualization: perception for design (3rd Ed.)*. Elsevier, 2013.

Kosslyn, Stephen M. *Graph design for the eye and mind*. Oxford University Press, 2006. The principles identified in human communication for good data visualisation design are based on his eight principles of effective graphics.
Segel, Edward, and Jeffrey Heer. Narrative visualization: Telling stories with data. *IEEE Transactions on*

Visualization and Computer Graphics, 16, no. 6 : 1139-1148, 2010.

Further reading

- Segel & Heer, 2010.

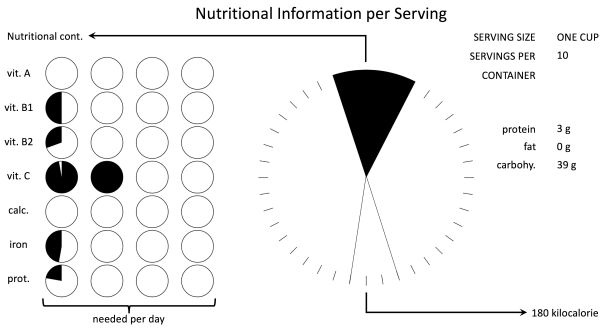
4
Activity: Effective graphic design

The phrase 'effective graphic design' begs the question 'effective for whom and for what?'.

Any graphic design must:

- Be usable by the user given her or his knowledge and abilities. Most people will be able to comprehend a common graphic such as a line graph or a table of data. However other forms of representation such as set diagrams (e.g. Euler's circles) are unfamiliar to many people, moreover some forms of representation are highly domain-specific e.g. cladograms (https://en.wikipedia.org/wiki/Cladogram) are pretty much exclusive to biology and are only familiar to specialists.
- Be suited to the task the user needs to perform with the representation. There are many kinds of tasks - a user might be searching, comparing, spotting an intermittent event (vigilance), seeing trends, making inferences or deductions, discovering, assembling furniture (https://www.tc.columbia.edu/faculty/bt2158/faculty-profile/files/plesforvisualization_Revealingandinstantiating.PDF) (e.g. Chapter 3 of Munster, also Amar & Stasko, 2004 (http://www.cs.kent.edu/~jmaletic/cs63903/papers/Amar04.pdf); Wehrend & Lewis, 1990 (http://dl.acm.org/citation.cfm?id=949553)).

So lets go back to the proposed nutritional display graphic



What I want you to do is think about

1. Who is going to be using this graphic and what is their background?
2. For what task(s) are they going to look at the graphic and what is the context?

Now based on the material you have read

1. Come up with three or four alternative designs which you think are better. One should be black-and-white, the others can use colour. You may want to use the Five Design Sheet methodology (https://www.alexandriarepository.org/module/activity-five-design-sheet-methodology/).
2. Evaluate the quality of these designs in terms of the principles given in Effective Communication and the perceptual and cognitive characteristics given in The Human Visual System. Rank them.

FURTHER READING

Munzner, T. (2014) Visualisation, analysis and design. CRC Press

Hegarty, M. (2011) The cognitive science of visual-spatial displays: Implication for design. Topics in Cognitive Science, 3, 446-474.
(http://onlinelibrary.wiley.com.ezproxy.lib.monash.edu.au/doi/10.1111/j.1756-8765.2011.01150.x/abstract)

Kosslyn, Stephen M. *Graph design for the eye and mind*. Oxford University Press, 2006.

5
Activity: Creating visualisations with D3

A. An Introduction to D3

D3 (**Data-Driven Documents)** is a HTML5/SVG + javaScript based data visualisation toolkit. It allows you to do dynamic and interactive data visualisations in web browsers. It is not really for exploratory data analysis (use Tableau, R or Python, or.....) but it is intended to publish data visualisations

D3 is extremely powerful, however, it's equally complicated. You can have a look at some of the examples:
https://github.com/mbostock/d3/wiki/Gallery
Or dive in and look at some code e.g. a Choropleth map:
http://bl.ocks.org/mbostock/4060606

Or there is a free comprehensive online book of D3:

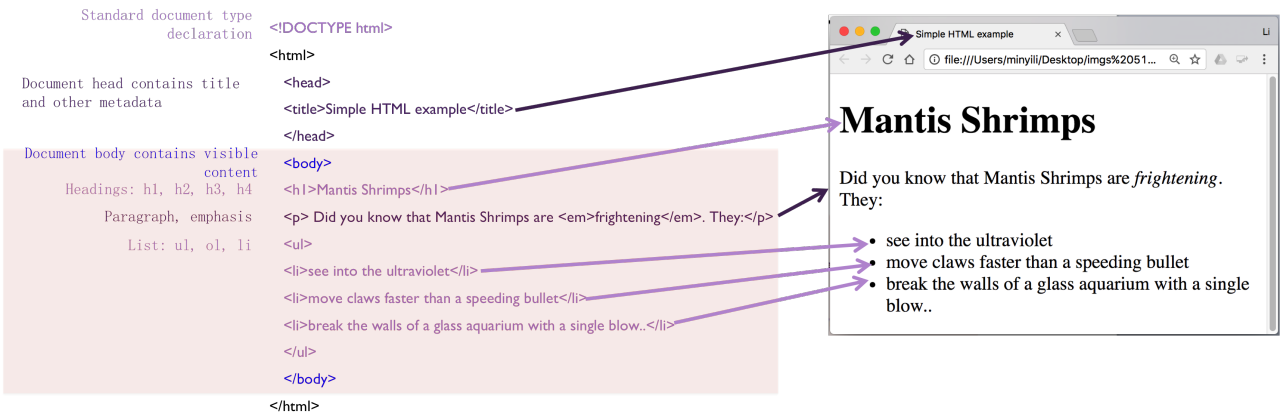
Interactive Data Visualization for the Web by Scott Murray
(http://chimera.labs.oreilly.com/books/1230000000345/index.html)

Working with D3 requires an appreciation of concepts including HTML, CSS, SVG (Scalable Vector Graphics), JavaScript and the DOM (Document Object Model). We will briefly review some of these in the following.

HTML

Hypertext Markup Language (HTML) is used to structure content for web browsers. HTML elements (represented by tags) are the building blocks of HTML pages. Following is an example of HTML, where the tags are annotated with its meaning on the left, and the visible page layout as on the right:

example.html



All HTML elements can be assigned attributes using property/value pairs in the opening tag. In the following example, the `href` attribute specifies the link's destination:

```
<a href="http://d3.js.org/">The D3 Website</a>
```

The class and id attribute can be used with any element. The id attribute specifies a **unique** id for an HTML element (i.e., the value must be unique within the HTML document). The class attribute specifies one or more **classnames** for an element.

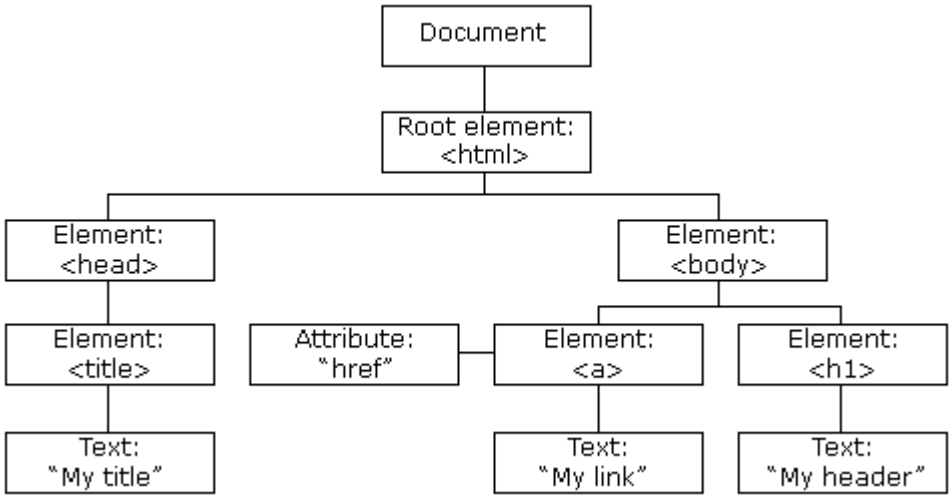
The id (resp. class) attribute is mostly used to point to a style (resp. a class) in a style sheet, and by JavaScript (via the HTML DOM) to manipulate the element with the specific id (resp. class).

Notice that an ID value can only be used once in a page. For instance, we can define the following two HTML elements of the same class "boring" but they must have different ids:

```
<p class="boring" id="p1"> In the beginning…</p>
<p class="boring" id="p2"> And so it goes on…</p>
```

Document Object Model (DOM)

The Document Object Model (DOM) is a cross-platform and language independent API. In HTML, it defines the hierarchical structure of the HTML document and so Web browsers can parse the DOM in order to make sense of a page's content.



From

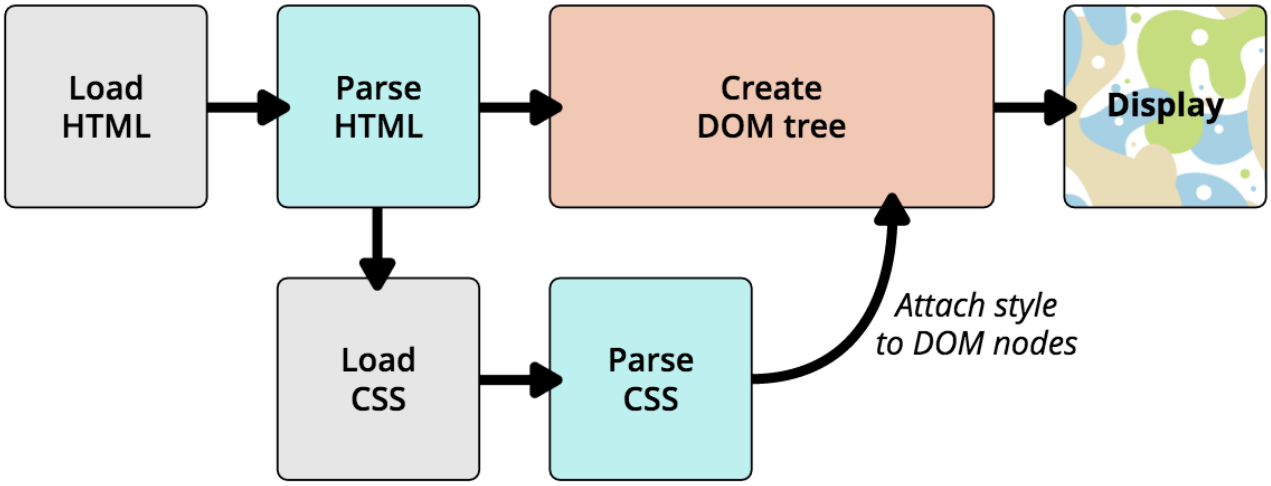
http://www.w3schools.com/js/js_htmldom.asp

In DOM, each bracketed tag is an *element*. Elements are defined as objects. Each element could have an relationship to another element, which could be expressed in human terms: parent, child, sibling, ancestor, and descendant. In the HTML above, `body` is the parent element to its children, `h1`, `p` and `ul`(which are siblings to each other). All elements on the page are descendants of `html`. HTML DOM also defines:

- ***Properties*** of all HTML elements;
- ***Methods*** to access all HTML elements; and
- ***Events*** for all HTML elements.

CSS

HTML aims to separate content from style, and it can use Cascading style sheets (CSS) to associate style information with HTML elements. CSS is a style sheet language that describes the visual presentation of an HTML document, it explains how HTML elements should be displayed.



From

https:// (https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works)developer.mozilla.org (https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works)/ (https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works)en (https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works)-US/docs/Learn/CSS/ (https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works)Introduction_to_CSS (https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works)/ (https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works)How_CSS_works (https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works)

CSS styles consist of rules made of selectors and properties. A simple CSS rule looks like this:

```
h1,p {
color: pink;
font-family: Arial;
}
```

In the above rule, `h1`, `p` are selectors; `color` and `font-family` are the properties of the rule, with values "pink" and "Arial" to specify its color and font famillly, respectively. If rules of different selectors have identical declarations, we can condense them into one. In the previous example, we've specified rules for `h1` and `p` elements in one go!

D3 uses CSS-style selectors to identify elements on which to operate, following are some example selectors:

Type selectors	h1 /* selects all level 1 headings*/
Descendent selectors:	p em /* selects all emphasized text in a paragraph*/
Class selectors:	.axis /* selects all elements with class axis*/ .axis.y /* selects all elements with class axis and class y */

ID selectors: #L1 /* selects element with ID "L1″*/

As you may have noticed, in some of the above examples, we've strung the selectors together, e.g., "p em" matches all **emphasized** text **in** a **paragraph** and ".axis.y". matches all elements with class axis **and** class y. For more complex examples of constructing CSS selectors, please refer to http://www.w3schools.com/cssref/css_selectors.asp

CSS rules can be used in-line in a HTML document, e.g.,`<p style = "color:blue">...</p>`

Or, they can be saved in an external file with a `.css` suffix, and then linked from the HTML head, like so:

```
<head>
    <link rel="stylesheet" href="style.css">
</head>
```

Also, they can be included directly within the document head using `<style>...</style>`

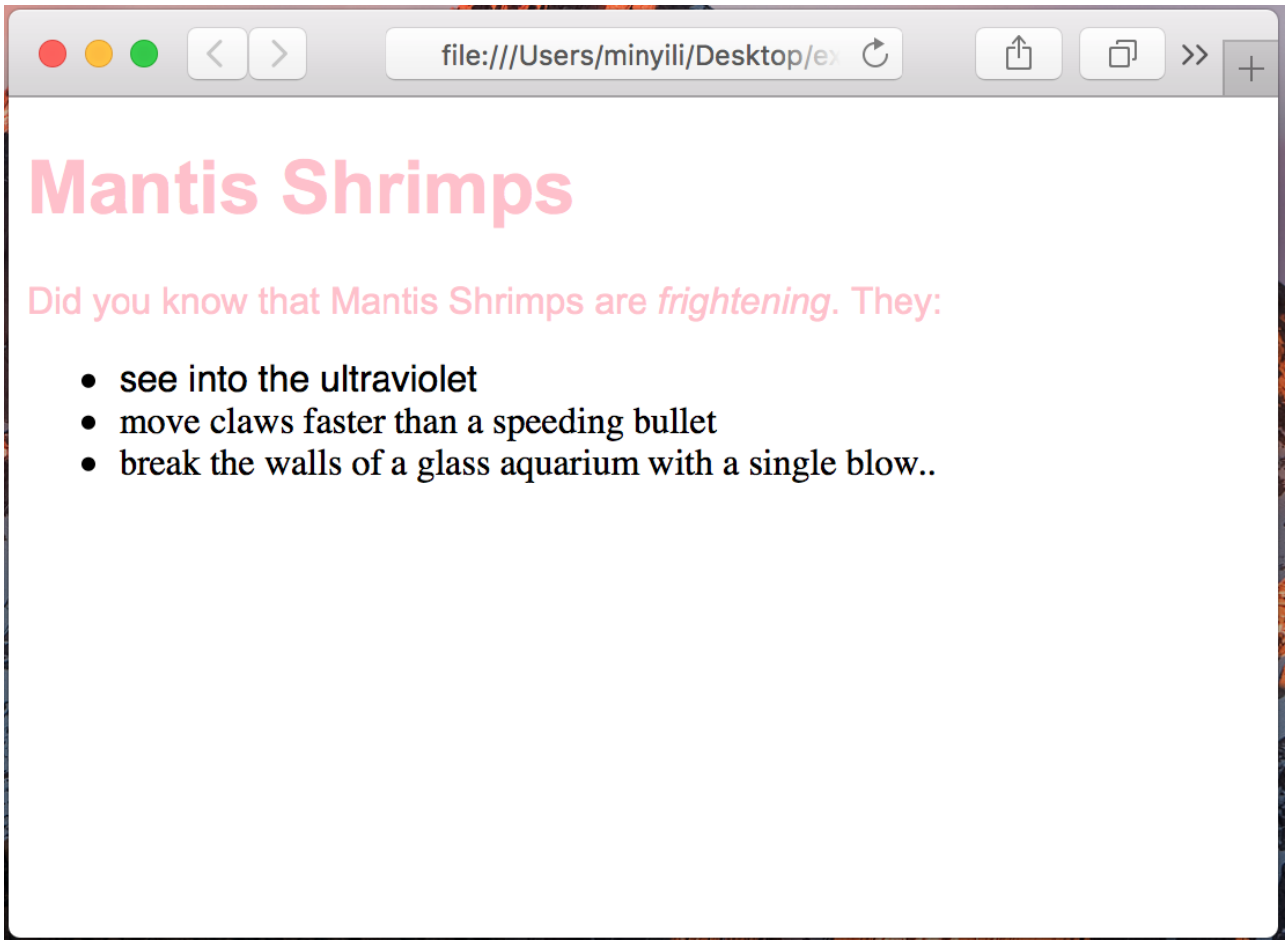
Following is an extended example of the previous HTML document:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Simple HTML example</title>
    <style type="text/css">
      h1,p {color: pink;  font-family: Arial;}
      #L1 {font-family: sans-serif; font-size: 16px;}
    </style>
  </head>
  <body>
  <h1>Mantis Shrimps</h1>
  <p> Did you know that Mantis Shrimps are <em>frightening</em>. They:</p>
  <ul>
    <li id="L1">see into the ultraviolet</li>
    <li>move claws faster than a speeding bullet</li>
    <li>break the walls of a glass aquarium with a single blow..</li>
  </ul>
  </body>
</html>
```

- The first CSS rule defines the styles for all the elements <h1> and <p> elements: using font-family

"Arial" and with color "pink".
- The second CSS rule specifies the style (font family and size) for elements with id "L1", which has been refer to later in one of the list element <li>.

You can see how these CSS rules affect the displays of the original HTML page:

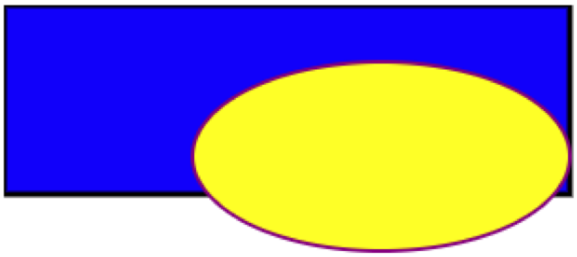


Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG) is the web vector graphics format. It is designed to work with both HTML and CSS, and so it can be directly included in the document.

A SVG canvas has width and a height and it contains standard graphic primitives: line, circle, text, ellipse, rect, path.

The coordinate system of SVG has origin (0,0) at top left corner, with the positive x-axis pointing towards the right, the positive y-axis pointing down, and one unit in the initial coordinate system equals one "pixel". Following is an example SVG, where a blue rectangle overlapped with a yellow ellipse is coded into a HTML page:



```
<!DOCTYPE html>
<html>
    <body>
        <svg width="400" height="110">
            <rect width="300" height="100" style="fill:rgb(0,0,255);stroke-
width:3;stroke:rgb(0,0,0)"/>
            <ellipse cx="200" cy="80" rx="100" ry="50"
style="fill:yellow;stroke:purple;stroke-width:2" />
        </svg>
    </body>
</html>
```

There are some common SVG styles, e.g.,"fill" paints the interior of the given graphical element; "stroke" paints along the outline of the given graphical element; "stroke-width" specifies the width of the stroke on the current object; etc.. For more SVG styles, please refer to https://www.w3.org/TR/SVG/styling.html

Javascript

JavaScript is a scripting language that can make web pages dynamic by modifying the DOM after the page has been loaded. It is a interpreted and dynamically typed programming language.

Following is examples of Javascript code:

```
//Dynamic, random dataset
var dataset = [];  //Initialize empty array
var numDataPoints = 50;  //Number of dummy data points to create
var xRange = Math.random() * 1000;  //Max range of new x values
var yRange = Math.random() * 1000;  //Max range of new y values
for (var i = 0; i &amp;amp;amp;lt; numDataPoints; i++)
    {  //Loop numDataPoints times
        var newNumber1 = Math.floor(Math.random() * xRange);
        var newNumber2 = Math.floor(Math.random() * yRange);
        dataset.push([newNumber1, newNumber2]);  //Add new number to array
    }
```

JavaScript can be stored in a separate file with .js suffix and referenced in the file, e.g.,

```
<head>
    <title>Page Title</title>
    <script type="text/javascript" src="myscript.js"></script>
</head>
```

Scripts can also be included directly in HTML, between two `script` tags, e.g.,

```
<body>
    <script type="text/javascript"> alert("Hello, world!") </script>
</body>
```

In the following, we briefly visit the very basics of JavaScript.

Arrays and Objects

Similar to other programming languages like Java or C++, JavaScript supports arrays. The following example declares an array of four integers:

```
var data = [ 5, 6, 7, 8];
```

Notice array elements in Javascript are accessed with index from 0 to length-1. Elements can have different type (Yuck!) and we can define multidimensional arrays, i.e., arrays of arrays. We can also assess the size of the array through `length` attribute, e.g., `data.length` returns value 4.

JavaScript also provides object. For example, we can define the follow "fruit" object, which contains three variables: kind, colour and quantity.

```
var fruit = {
              kind: "grape",
              colour: "red",
              quantity: 12
             };
```

We can refer to the variable values of an object using "." notation, e.g., `fruit.kind` will return value "grape".

We can also specify arrays of objects in JavaScript.

Operators & Control Structures

Following are some common operators and control structures applied in JavaScript:

Standard mathematical operators	`+ - * /`
Standard comparison operators	`== != < > <= >=`
For loops	`for (var i=0; i<5; i++) { console.log(i);}`
If-then-else	`if (i==5) { console.log("i=5")}` `else {console.log("i ne 5")};`

Functions

In JavaScript, functions group together script code; control structures, operations, method calls, etc. to perform a particular task. A JavaScript function is executed when "something" invokes it (i.e., calls it).

The two most common ways to create a function in javascript are by using the function declaration or

function operator. Following is a simple example of a declaring a user defined JavaScript **function** double:

```
function double(x) { return 2*x;}
```

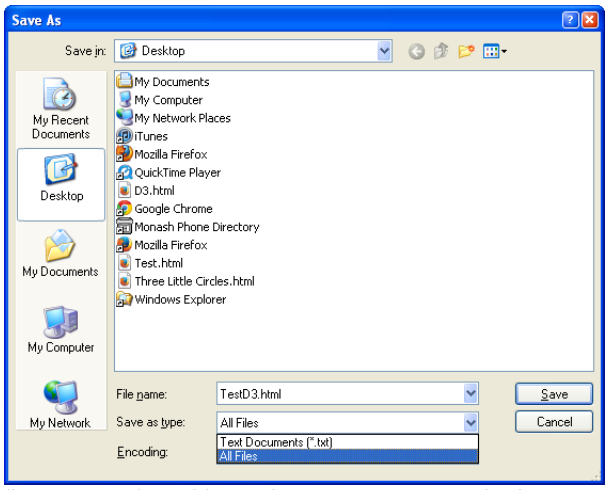
Also, you can pass anonymous functions as arguments (Anonymous functions are functions that are dynamically declared at runtime. They're called anonymous functions because they aren't given a name in the same way as normal functions.) Anonymous functions are created using the function operator. The following example creates an anonymous function and then assigns it to a **variable** called double.

```
var double= function(x) { return 2*x; }
```

Embedded within an HTML webpage, the JavaScript D3.js library uses pre-built JavaScript functions to create, select, and control the dynamic behavior of the HTML elements and SVG objects. These objects and elements can be widely styled using CSS. For more details about the fundamentals of D3, please refer to http://alignedleft.com/tutorials/d3/fundamentals. The D3 site (http://d3js.org) (or this article (http://vis.stanford.edu/files/2011-D3-InfoVis.pdf)) also provides a starting point to learn about D3.

Now we are ready to try some of examples of D3. We will begin with one of the tutorials from the creator of D3, Mike Bostock (http://bost.ocks.org/mike/circles/) but with some gaps filled in and a few changes. If you have a preferred editor use that, otherwise all you need is Notepad (or maybe Wordpad, but not MS Word) and a browser (Firefox here, or Chrome)... on with the show.

Step 1
Create a new file, call it e.g. 'TestD3.html' and save (be careful about saving as 'All files' in Notepad, not .txt, you may get '.html.txt')



(https://www.alexandriarepository.org/wp-content/uploads/20150929033657/D3-1.png)

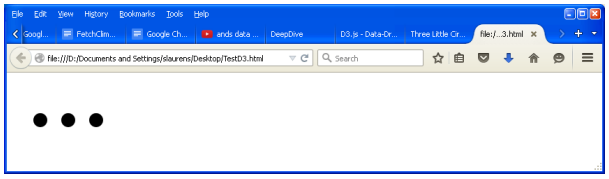
Step 2
Type or paste the following html

```
<svg width="720" height="120">
  <circle cx="40" cy="60" r="10"></circle>
  <circle cx="80" cy="60" r="10"></circle>
  <circle cx="120" cy="60" r="10"></circle>
</svg>
```

Where a circle is defined by x, y locations (cx, cy) and radius (r) which are all the same (as are the cy) so

the circles are all the same size and in a line.

Step 3
Save (as html), now open the file with your browser to see:



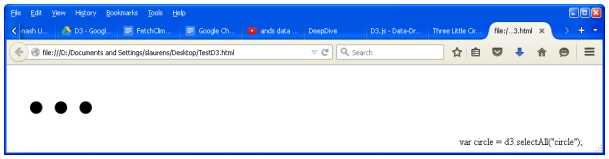
Three little circles... these are not images per se (try to click on them or try to 'save as'), they are scalable vector graphics, SVG is built into (most) browsers. Use CTRL + and CTRL - to zoom, note the smooth shapes, this is the scalable and vector aspects at work. We can change them using D3 too. This is the essence of D3, change elements in the browser, let the browser draw them.

Step 4
Back in Notepad, add this javascript code below the svg HTML:

```
var circle = d3.selectAll("circle");  // select all circles
circle.style("fill", "red");  // change colour
circle.attr("r", 30);  // change radius
```

What do you expect to see?

Step 5
Save and reload, oops, no changes, the code is being displayed as text (below right)...



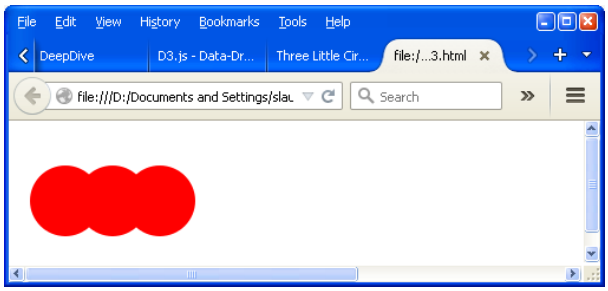
Step 6
Add the <script></script> tags around the javascript, save and reload, i.e.,:

```
<script>
var circle = d3.selectAll("circle"); // select All circles
circle.style("fill", "red"); // change colour
circle.attr("r", 30); // change radius
</script>
```

Step 7
Tidier but still not working, we have to connect to D3.js so add this line at the top:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.5/d3.min.js"
charset="utf-8">
</script>
```

(you can download and use a local copy of D3 but this is easier).
Save and open (or reload/refresh) in your browser:



Note that if you have any errors in your javascript then it will **just not run**.
Try this: accidentally leave an extra bracket in your code:

```
<script>
var circle = d3.selectAll("circle");  // select All circles
circle.style("fill", "red");  // change colour
circle.attr("r", 30);  // change radius
} // oops
</script>
```

You may expect to see 3 red circles but all you get is the original svg elements - beware. (and delete the error to continue).

Step 8
Now right click then 'View page source' to see:

view-source:file://ad.monash.edu/home/User058/slaurens/Desktop/D3Test.html

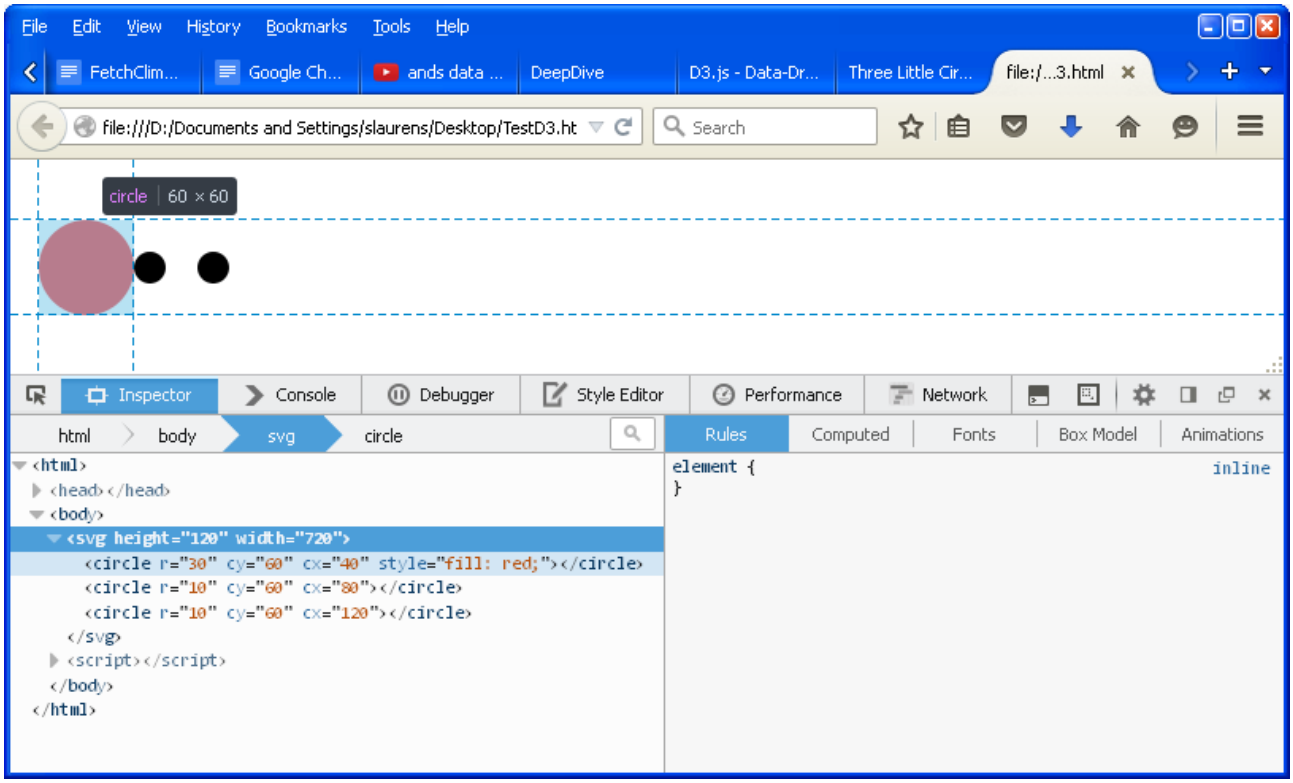
```
1  <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.5/d3.min.js" charset="utf-8"></script>
2
3  <svg width="720" height="120">
4          <circle cx="40" cy="60" r="10"></circle>
5          <circle cx="80" cy="60" r="10"></circle>
6          <circle cx="120" cy="60" r="10"></circle>
7  </svg>
8
9  <script>
10         var circle = d3.selectAll("circle");    // select All circles
11         circle.style("fill", "red");            // change colour
12         circle.attr("r", 30);                   // change radius
13 </script>
```

(Note that this is minimal html, no,etc. tags, no adwords...)

Step 9
Change the line from 'selectAll' to 'select', save and run:

Step 10

Let's see what D3 is doing in the background. Right click in your browser and 'Inspect Element' (note that this is FireFox, same command in Chrome, slightly different view). Then open the 'svg' element (below left) to see:



We used D3 to create a new DOM element, which the browser then renders.
Note "fill: red" and the radius (r="30") - the HTML has been changed by the code. Close the lower frame using 'x' (far right) when you're finished inspecting.

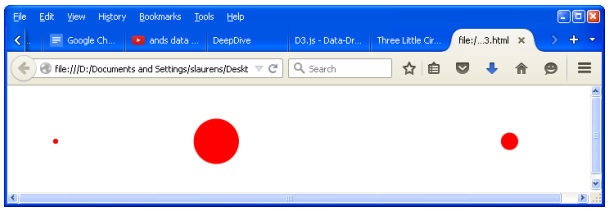
Step 11

We've seen 'selectAll' and first element only (the default for select), try doing something to each element. Change back to 'selectAll' then add these two lines:

```
circle.attr("cx", function() { return Math.random() * 720; });  // function
returns a random location on the x axis,
// see Step 2 above, x is up to 720, .attr applies it to "cx" (should be a
constant...)
circle.attr("r", function() { return Math.random() * 50; });
// random size function
```

Step 12

And run (note that the circles *could* all end up the same size in the same location).
Run/refresh a few times to see that the location and size are random e.g.



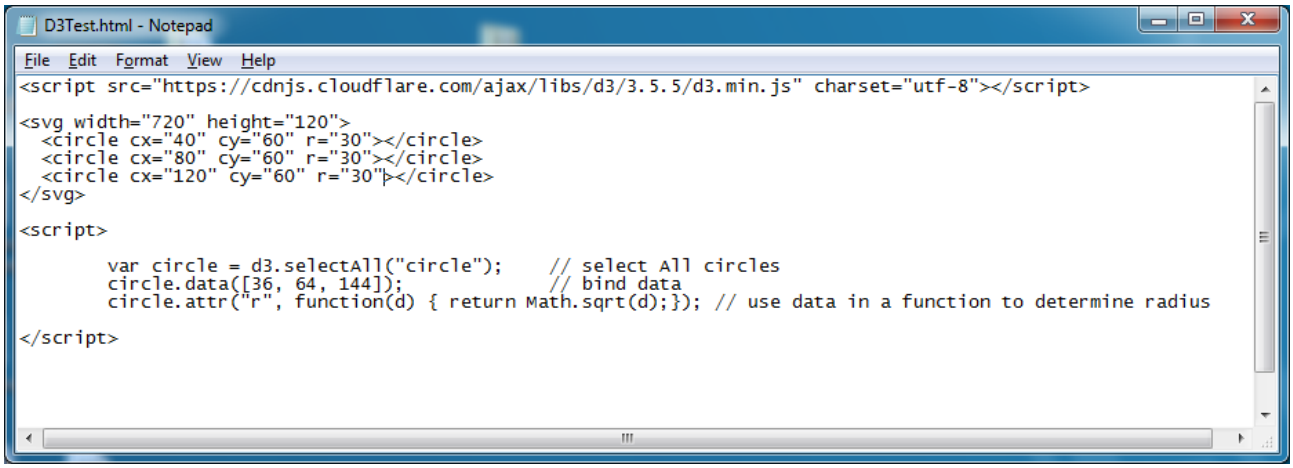
Step 13

Binding data - e.g. use *data* to change the appearance of our circles, we can hard code the data (below, but hopefully we can have files or live data eventually...).

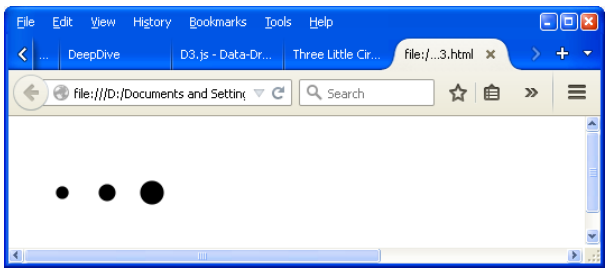
```
circle.data([36, 64, 144]);
circle.attr("r", function(d) { return Math.sqrt(d); });
```

The binding is in order (first circle:36, second:64 etc.). Note that the '36' doesn't mean anything yet (just like Bob:185 gains meaning only *after* we say "Bob is 185cm tall").

Comment out lines (or delete them) to get:



And run to see:



What values for radius do you expect to find in the DOM?

Step 14

Change the function to multiply by e.g. 2, one of these will work, one won't:

```
circle.attr("r", function(d) { return Math.sqrt(d) * 2; });
circle.attr("r", function(d) { return Math.sqrt(d); * 2});
```

Which of the above worked, which didn't, and why?

And make random colours too:

```
circle.style("fill",function() { return "hsl(" + Math.random() * 360 +
",100%,50%)"; })
```

What is hsl?

As before, run this a few times to see changes e.g.

So that's an example to D3, DOM, SVG, (and some HTML but no CSS yet)

B. Charting with D3, Bar charts

Step 1
Start a new file in Notepad, e.g. D3BarChart.html, add the reference to D3.js e.g.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.5/d3.min.js"
charset="utf-8"></script>
```

Decide on the size of the new SVG:

```
//Width and height
var w = 500;
var h = 100;
```

Then, tell D3 to create an empty SVG element and add it to the DOM:

```
//Create SVG element
 var svg = d3.select("body")
             .append("svg")
             .attr("width", w)
             .attr("height", h);
```

Next, generate rects and add them to svg.

```
svg.selectAll("rect")
   .data(dataset)
   .enter()
   .append("rect")
   .attr("x", 0)
   .attr("y", 0)
   .attr("width", 20)
   .attr("height", 100);
```

Some pretend data:

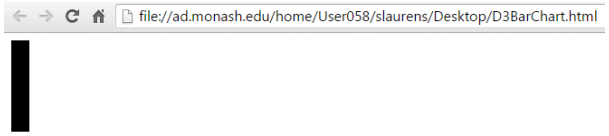
```
var dataset = [ 5, 10, 13, 19, 21, 25, 22, 18, 15, 13];
```

Step 2

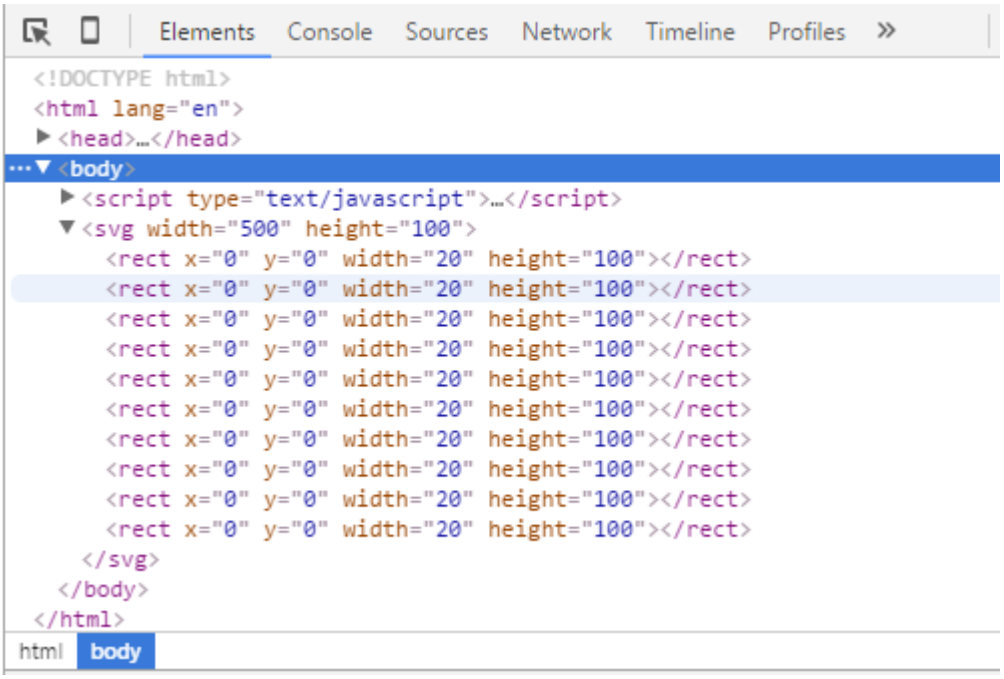
Put it all together (in slightly more proper HTML):

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.5/d3.min.js"
charset="utf-8">
        </script>
    </head>
    <body>

        <script type="text/javascript">
            //Width and height
            var w = 500;
            var h = 100;
            var dataset = [ 5, 10, 13, 19, 21, 25, 22, 18, 15, 13];

            //Create SVG elements
            var svg = d3.select("body")
                        .append("svg")
                        .attr("width", w)
                        .attr("height", h);
            svg.selectAll("rect")
               .data(dataset)
               .enter()
               .append("rect")
               .attr("x", 0)
               .attr("y", 0)
               .attr("width", 20)
               .attr("height", 100);
        </script>
    </body>
</html>
```

Save, load and browse…



All of the bars are there (check the DOM in your web inspector, shown below), but they all share the same x, y, width, and height values, so they all overlap…



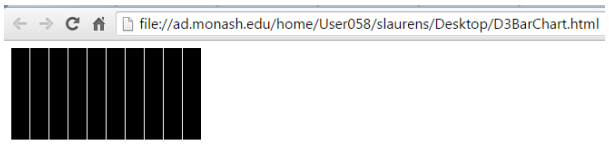
Step 3

Let's fix the overlap issue first. Assign different values for each position in the data set. The first bar will be at zero, but subsequent bars will be at 21, then 42, and so on.

```
.attr("x", function(d, i) {
    return i * 21;  // Bar width of 20 plus 1 for padding
})
```

But where does this function go in the code?

(replace the .attr("x", 0) line with the function)



(https://www.alexandriarepository.org/wp-content/uploads/20150929033657/D3-14.png)

That works, but it's not particularly flexible. If our data set were bigger, then the bars would just run off to the right, past the end of the SVG. So scale based on this size.

Step 4

```
.attr("x", function(d, i) {
    return i * (w / dataset.length);
})
```

Now we should set the bar *widths* to be proportional too, so they get narrower as more data is added, or wider when there are fewer values.

Add a constant/variable to the others (width and height)

```
var barPadding = 1;
```

and then reference that it in the line where we set each bar's width. Instead of a static value of 20, the width will now be set as a fraction of the SVG width and number of data points, minus a padding value:

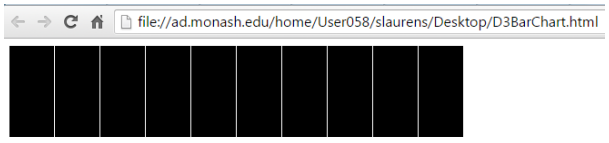
```
.attr("width", w / dataset.length - barPadding)
```

But where does this go in the code?

The script snippet should now look like this:

```
svg.selectAll("rect")
   .data(dataset)
   .enter()
   .append("rect")
   .attr("x", function(d, i) { return i * (w / dataset.length); })
   .attr("y", 0)
   .attr("width", w / dataset.length - barPadding)
   .attr("height", 100);
```

And generate this:



(https://www.alexandriarepository.org/wp-content/uploads/20150929033657/D3-15.png)

Step 5
Double the dataset (randomly or cut and paste the 10 points), save, run...

Crank it up to 100 if you like... (what would happen at 200? at 500?)

Note that the values in the dataset are having no effect on the chart yet because all heights are 100. Time for a change:

Step 6

```
.attr("height", function(d) { return d; });
```

Note that 'd' in D3 is a naming convention, it refers to the binding to the data (otherwise this would appear to be rather magical).
Each 'd' in data is bound to each element, D3 does them all for you (in order), no need for a loop.

Save, load and browse:

oops…
What does this tell you about SVG?

Given that our bars do have to "grow down from the top," then where is "the top" of each bar in relationship to the top of the SVG? Well, the top of each bar could be expressed as a relationship between the height of the SVG and the corresponding data value, as in:

```
.attr("y", function(d) {
    return h - d;  // Height minus data value
})
```

Then, to put the "bottom" of the bar on the bottom of the SVG, each rects height can be just the data value itself:

```
.attr("height", function(d) {
    return d;  //Just the data value
});
```

Step 7
So now the relevant script section looks like this:

```
svg.selectAll("rect")
   .data(dataset)
   .enter()
   .append("rect")
   .attr("x", function(d, i) { return i * (w / dataset.length); })
   .attr("y", function(d) { return h - (d * 4); })
   .attr("width", w / dataset.length - barPadding)
   .attr("height", function(d) { return d * 5; }); // make them a bit bigger
```

Step 8 Colour
Adding color is easy. Just use attr() to set a fill:

```
.attr("fill", "blue");
```

or

```
.attr("fill", function(d) {
    return "rgb(0, 0, " + (d * 10) + ")";
});
```

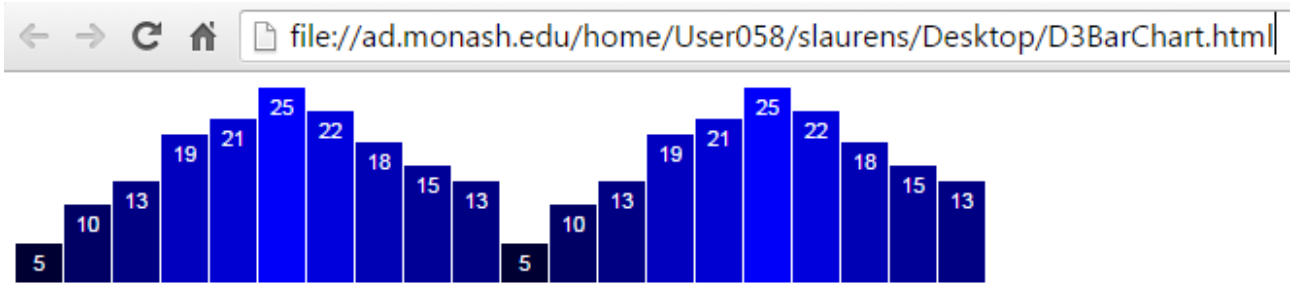
Step 9 Labels

Visuals are great, but sometimes you need to show the actual data values as text within the visualisation.

Here's where value labels come in, and they are easy to generate with D3.

This is similar to what's been done with rects & attrs above so here it all is in one go:

```
svg.selectAll("text")
   .data(dataset)
   .enter()
   .append("text")
   .text(function(d) { return d; })
   .attr("text-anchor", "middle")
   .attr("x", function(d, i)
                        { return i * (w / dataset.length) + (w / dataset.length -
barPadding) / 2; })
   .attr("y", function(d) { return h - (d * 4) + 14; })
   .attr("font-family", "sans-serif")
   .attr("font-size", "11px")
   .attr("fill", "white");
```



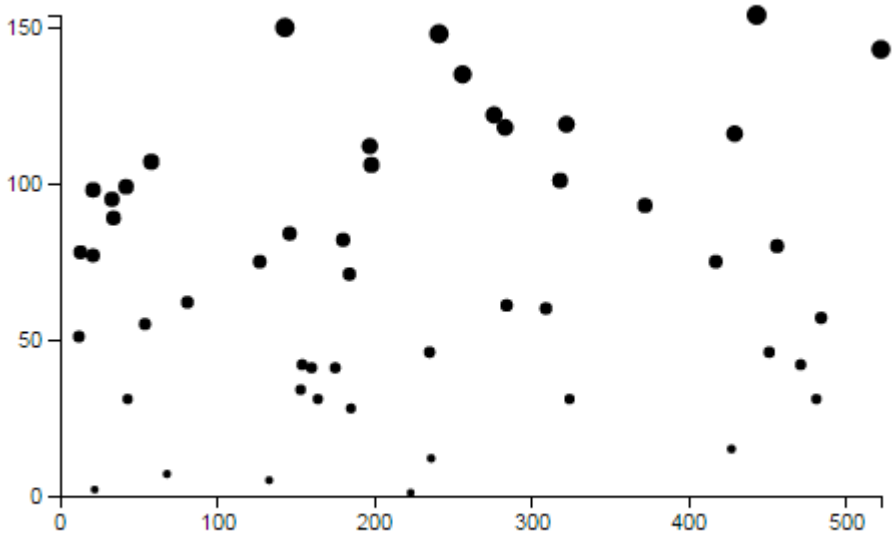
And that's a decent looking bar chart, based on:
http://alignedleft.com/tutorials/d3

C. Charting with D3, Scatterplots

Step 1
Create a new file (in Notepad), e.g. D3ScatterBasic.html and run the following:

```
<meta charset="utf-8">
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>D3 Demo: Making a scatterplot with SVG</title>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.5/d3.min.js"
charset="utf-8">
        </script>
    </head>
    <body>
        <script type="text/javascript">
            //Width and height
            var w = 500;
```

```
            var h = 100;
            var dataset = [ [5, 20], [480, 90], [250, 50], [100, 33], [330, 95],
[410, 12], [475, 44], [25, 67], [85, 21], [220, 88]];
            //Create SVG element
            var svg = d3.select("body")
                        .append("svg")
                        .attr("width", w)
                        .attr("height", h);

            svg.selectAll("circle")
               .data(dataset)
               .enter()
               .append("circle")
               .attr("cx", function(d) { return d[0]; })
               .attr("cy", function(d) { return d[1]; })
               .attr("r", 5);
        </script>
    </body>
</html>
```



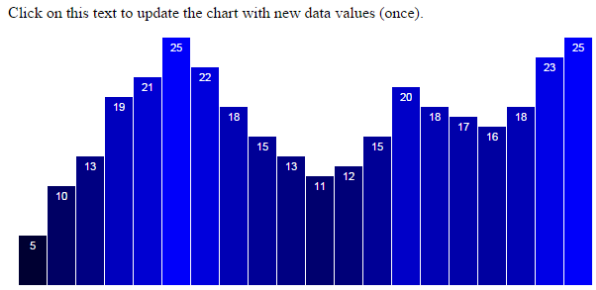
Step 2
Here's a more complete example, note the use of CSS (typically in a separate file), the dynamic (random) data, scales & axes (in the code). Save it, run it, view source: D3Scatter.html
(https://www.alexandriarepository.org/wp-content/uploads/20150929033657/D3Scatter.html)

based on http://alignedleft.com/tutorials/d3

D. Transitions (AKA animation)

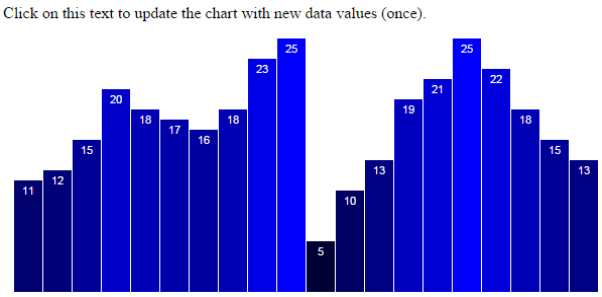
And finally some animation, save it, run it, click on the text to see the transition, view source:

D3Transition.html (https://www.alexandriarepository.org/wp-content/uploads/20150929033657/D3Transition.html)



(https://www.alexandriarepository.org/wp-content/uploads/20150929033657/D3-20.png)

Refresh/reload to reuse



From the book 'Interactive Data Visualization for the Web'
http://examples.oreilly.com/0636920026938/chapter_09/05_transition.html