

COMP5048 Week 3 Tutorial

1. Introduction

D3.js is a Javascript library for data manipulation and visualization. It provides convenient functions to modify DOM elements based on data, allowing the creation of complex visualisations. Downloads for the source code and examples can be found from d3js.org.

This tutorial will focus on the basics of data manipulation with D3 and examples of tree visualisations using D3.

2. Running examples

Some of the examples need to be run on a server due to the need to load other files from the filesystem. Note: if you have a server running on your own machine, feel free to use that instead.

1. Extract *comp5048-wk3.zip* to a folder.
2. Open the command line and navigate to the destination of extraction.
3. Run `python -m SimpleHTTPServer 8888` from the command line.
4. The example web pages can now be accessed through localhost; that is, *localhost:8888/treert.html*.

3. Examples runthrough

3.1 Selections

1. Open *text.html* in a browser (Firefox / Chrome recommended) and open the developer's console (ctrl+shift+i).
2. Inspect the source ("Element" tab from console). Note that the body contains an *h1* object, an *h2* object, and two *p* objects.
3. In the console, type `d3.select("h1").text("Level 1 heading")`. The first heading will now have its text changed
4. Do a similar operation by selecting *p* instead: `d3.select("p").text("Paragraph")`. You will notice that only the first *p* object is affected.
5. To select and modify all objects of the same type, use the *selectAll* function: `d3.selectAll("p").text("This is a paragraph")`.

3.2 Simple drawing

1. Open the page at *localhost:8888/circles.html*.
2. Inspect the source and take note of the pre-defined variables.
3. In the developer's console, type `var svg = d3.select("body").append("svg")`, which creates a new (empty) *svg* tag inside the *body* tag.
4. Create a selection of circles contained by the SVG: `var circles = svg.selectAll("circle")`. This selection is actually empty as there are no circle items yet within the SVG.
5. To draw circles centred at the coordinates defined in the *centres* array, attach it to the *circles* selection and "enter" the selection of objects defined by the data:
`circles = circles.data(centres).enter()`.

The selection now contain two objects corresponding to the entries of the *centres* array, which is now attached as the data array of the selection.

6. Create circles based on the selection: `circles = circles.append("circle")`. This creates two circle objects (not yet drawn) corresponding to each entry in the data array.

7. To define the centre coordinates based on the information in the data array, define the *cx* and *cy* attributes of the circles:

```
circles.attr("cx", function(d) { return d.x; } ).attr("cy", function(d){ return d.y; } ).
```

Here, *d* represents an entry in the data array, and the function defined is run once for each entry.

8. The radius of the circle is defined using the "r" attribute: *circles.attr("r", 5).*

9. To define the appearance of the circles, use the style function. For example, to colour the circles red, type *circles.style("fill", "red")*. Alternatively, the style can be defined beforehand using CSS.

Note: While this example goes through the function calls one-by-one for clarity, a more compact way is by method chaining all the methods executed on the same selection, e.g. *circles.append("circle").attr("cx", function(d) { return d.x; }).attr("cy", function(d) { return d.y; }).attr("r", 5).* For an example, see *circles-worked.html*.

3.3 Tree layouts

This example has been modified from <http://bl.ocks.org/mbostock/4339184>.

1. Open *tree-rt.html* in a browser to see what the visualisation looks like.
2. Examine the page source. Here, the data is read from an external JSON file in the *d3.json* function. The entries in the JSON file do not have coordinates defined - these are defined by the layout function (the variable *tree*, containing D3's tree layout *d3.layout.tree*).
3. In the developer's console, print out the contents of the nodes and links, e.g. *console.log(nodes)*. The former is an array of the nodes of the tree, now with "x" and "y" coordinates defined, while the latter defines the edges within the tree, with each entry containing a "source" and "target".
4. The edges are drawn by taking the links array as the data, and adding path objects per entry. The lines are drawn using a generator, which creates a path based on an array of objects/points. Here, a diagonal generator is used, which takes as input an array of objects with "source" and "target" attributes.
5. The nodes are drawn by taking the nodes array as the data and adding circle objects, although in a slightly different fashion to the previous example: the *cx* and *cy* attributes are left as their default value of 0, and instead the circle is drawn inside a group (*g* object) that is translated to the centre of the circle using the transform attribute.
6. *tree-linear.html* is a variation that uses straight lines, while *tree-radial.html* (from <http://bl.ocks.org/mbostock/4063550>) uses a radial tree layout.