Comp 424 - Client-side Web Design

Fall Semester 2016 - Week 13

Dr Nick Hayward

Contents

- Data storage
  - *MongoDB*
- Data visualisation
  - *intro*
  - *types*
- Data visualisation library - D3.js
  - *intro*
  - *data*
  - *selections*

Server-side considerations - data storage

MongoDB - test app

- now create a new test app for use with MongoDB
- create and setup app as before
  - *eg: same setup pattern as Redis test app*
- add **Mongoose** to our app
  - *use to connect to MongoDB*
  - *helps us create a schema for working with DB*
- update our `package.json` file
  - *add dependency for Mongoose*

```
// add mongoose to app and save dependency to package.json
npm install mongoose --save
```

- test server and app as usual from app's working directory

```
node server.js
```

Server-side considerations - data storage

MongoDB - Mongoose schema

- use **Mongoose** as a type of bridge between Node.js and MongoDB
- works as a client for MongoDB from Node.js applications
- serves as a useful data modeling tool
  - *represent our documents as objects in the application*
- a data model
  - *object representation of a document collection within data store*
  - *helps specify required fields for each collection's document*
  - *known as a schema in Mongoose, eg: `NoteSchema`*

```
var NoteSchema = mongoose.Schema({
  "created": Date,
  "note": String
});
```

- using schema, build a model
  - *by convention, use first letter uppercase for name of data model object*

```
var Note = mongoose.model("Note", NoteSchema);
```

- now start creating objects of this model type using JavaScript

```
var funchalNote = new Note({
"created": "2015-10-12T00:00:00Z",
"note": "Curral das Freiras..."
});
```

- then use the Mongoose object to interact with the MongoDB
  - *using functions such as `save` and `find`*

Server-side considerations - data storage

MongoDB - test app

- with our new DB setup, our schema created
  - *now start to add notes to our DB, `424db1`, in MongoDB*
- in our `server.js` file
  - *need to connect Mongoose to `424db1` in MongoDB*
  - *define our schema for our notes*
  - *then model a note*
  - *use model to create a note for saving to `424db1`*

```
...
//connect to 424db1 DB in MongoDB
mongoose.connect('mongodb://localhost/424db1');
//define Mongoose schema for notes
var NoteSchema = mongoose.Schema({
  "created": Date,
  "note": String
});
//model note
var Note = mongoose.model("Note", NoteSchema);
...
```

Server-side considerations - data storage

MongoDB - test app

- then update app's `post` route to save note to `424db1`

```
//json post route - update for MongoDB
jsonApp.post("/notes", function(req, res) {
  var newNote = new Note({
    "created":req.body.created,
    "note":req.body.note
  });
  newNote.save(function (error, result) {
    if (error !== null) {
      console.log(error);
      res.send("error reported");
    } else {
      Note.find({}, function (error, result) {
        res.json(result);
      })
    }
  });
});
```

Server-side considerations - data storage

MongoDB - test app

- update our app's `get` route for serving these notes

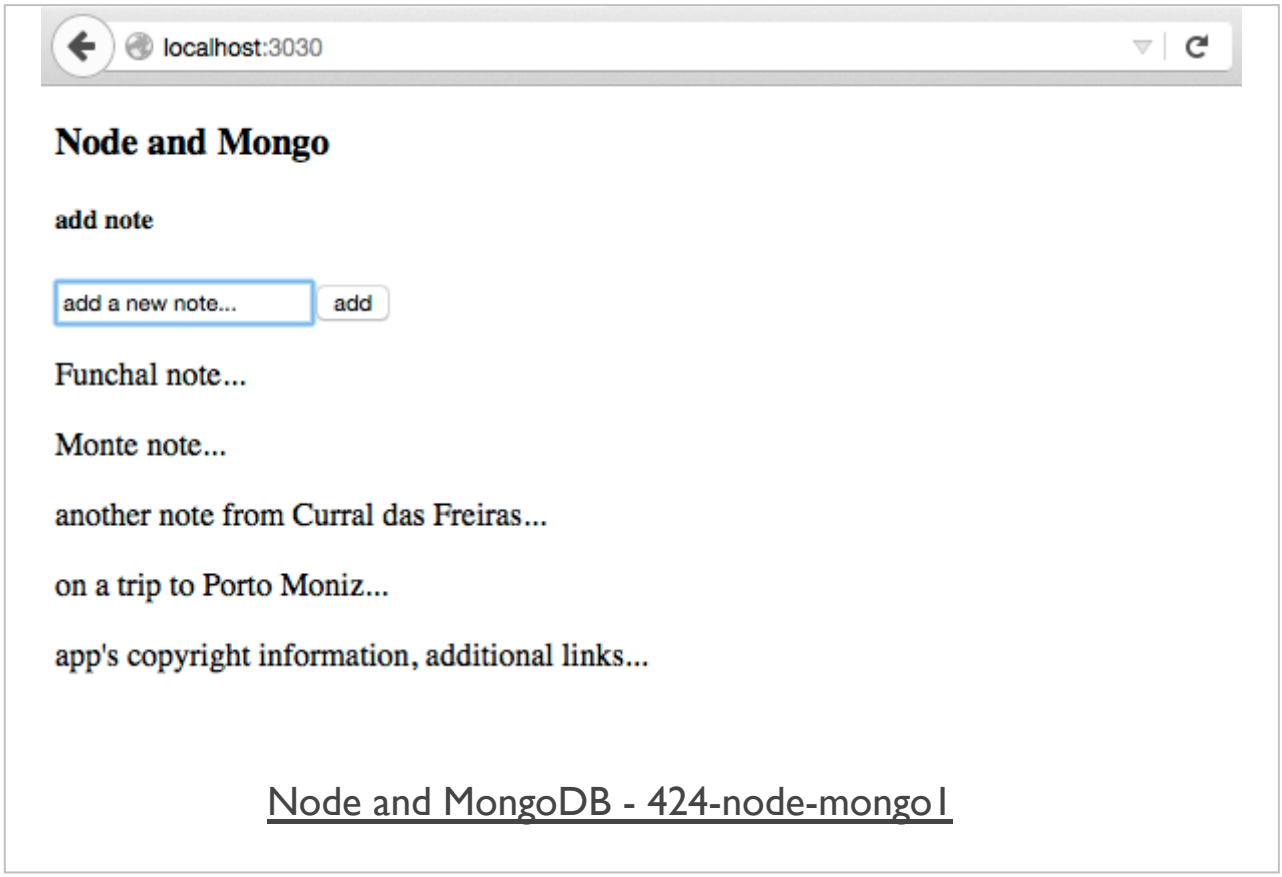
```
//json get route - update for mongo
jsonApp.get("/notes.json", function(req, res) {
  Note.find({}, function (error, notes) {
   //add some error checking...
   res.json(notes);
  });
});
```

- modify `buildNotes()` function in `json_app.js` to get return correctly

```
...
//get travelNotes
var $travelNotes = response;
...
```

- now able to enter, save, read notes for app
- notes data is stored in the `424db1` database in MongoDB
- notes are loaded from DB on page load
- notes are updated from DB for each new note addition
- DEMO - 424-node-mongo1

Image - Client-side and server-side computing



Node and MongoDB - 424-node-mongo1

Data visualisation

intro - part 1

- data visualisation - study of how to visually communicate and analyse data
- covers many disparate aspects
  - *including infographics, exploratory tools, dashboards...*
- already some notable definitions of data visualisation
- one of the better known examples,

> *"Data visualisation is the representation and presentation of data that exploits our visual perception in order to amplify cognition."*
>
> *(Kirk, A. "Data Visualisation: A successful design process." Packt Publishing. 2012.)*

- several variants of this general theme exist
  - *the underlying premise remains the same*
- simply, data visualisation is a visual representation of the underlying data
- visualisation aims to impart a better understanding of this data
  - *by association, its relevant context*

Data visualisation

intro - part 2

- an inherent flip-side to data visualisation
- without a correct understanding of its application
  - *it can simply impart a false perception, and understanding, on the dataset*
- run the risk of creating many examples of standard **areal unit** problem
  - *perception often based on creator's base standard and potential bias*
- inherently good at seeing what we want to see
- without due care and attention visualisations may provide false summations of the data

Data visualisation

types - part 1

- many different ways to visualise datasets
  - *many ways to customise a standard infographic*
- some standard examples that allow us to consider the nature of visualisations
  - *infographics*
  - *exploratory visualisations*
  - *dashboards*
- perceived that data visualisation is simply a variation between
  - *infographics, exploratory tools, charts, and some data art*

1. infographics
   - *well suited for representing large datasets of contextual information*
   - *often used in projects more inclined to exploratory data analysis,*
   - *tend to be more interactive for the user*
   - *data science can perceive infographics as improper data visualisation because*
   - *they are designed to guide a user through a story*
   - *the main facts are often already highlighted*
   - ***NB:** such classifications often still only provide tangible reference points*

Data visualisation

types - part 2

2. exploratory visualisations

- *more interested in the provision of tools to explore and interpret datasets*
- *visualisations can be represented either static or interactive*
- *from a user perspective these charts can be viewed*
- *either carefully*
- *simply become interactive representations*
- *both perspectives help a user discover new and interesting concepts*
- *interactivity may include*
- *option for the user to filter the dataset*
- *interact with the visualisation via manipulation of the data*
- *modify the resultant information represented from the data*
- *often perceived as more objective and data oriented than other forms*

3. dashboards

- *dense displays of charts*
- *represent and understand a given issue, domain...*
- *as quickly and effectively as possible*
- *examples of dashboards*
- *display of server logs, website users, business data...*

Data visualisation

Dashboards - intro

- dashboards are dense displays of charts
- allow us to represent and understand the key **metrics** of a given issue
  - *as quickly and effective as possible*
  - *eg: consider display of server logs, website users, and business data...*
- one definition of a dashboard is as follows,

> *"A dashboard is a visual display of the most important information needed to achieve one or more objective; consolidated and arranged on a single screen so the information can be monitored at a glance."*
>
> *Few, Stephen. Information Dashboard Design: The Effective Visual Communication of Data. O'Reilly Media. 2006.*

- dashboards are visual displays of information
  - *can contain text elements*
  - *primarily a visual display of data rendered as meaningful information*

Data visualisation

Dashboards - intro

- information needs to be consumed quickly
- often simply no available time to read long annotations or repeatedly click controls
- information needs to be visible, and ready to be consumed
- dashboards are normally presented as a complementary environment
- an option to other tools and analytical/exploratory options
- design issues presented by dashboards include effective distribution of available space
- compact charts that permit quick data retrieval are normally preferred
- dashboards should be designed with a purpose in mind
- generalised information within a dashboard is rarely useful
- display most important information necessary to achieve their defined purpose
- a dashboard becomes a central view for collated data
- represented as meaningful information

Data visualisation

Dashboards - good practices

- to help promote our information
  - *need to design the dashboard to fully exploit available screen space*
- need to use this space to help users absorb as much information as possible
- some visual elements more easily perceived and absorbed by users than others
- some naturally convey and communicate information more effectively than others
- such attributes are known as **pre-attentive attributes of visual perception**
- for example,
  - *colour*
  - *form*
  - *position*

Data visualisation

Dashboards - visual perception

- pre-attentive attributes of visual perception

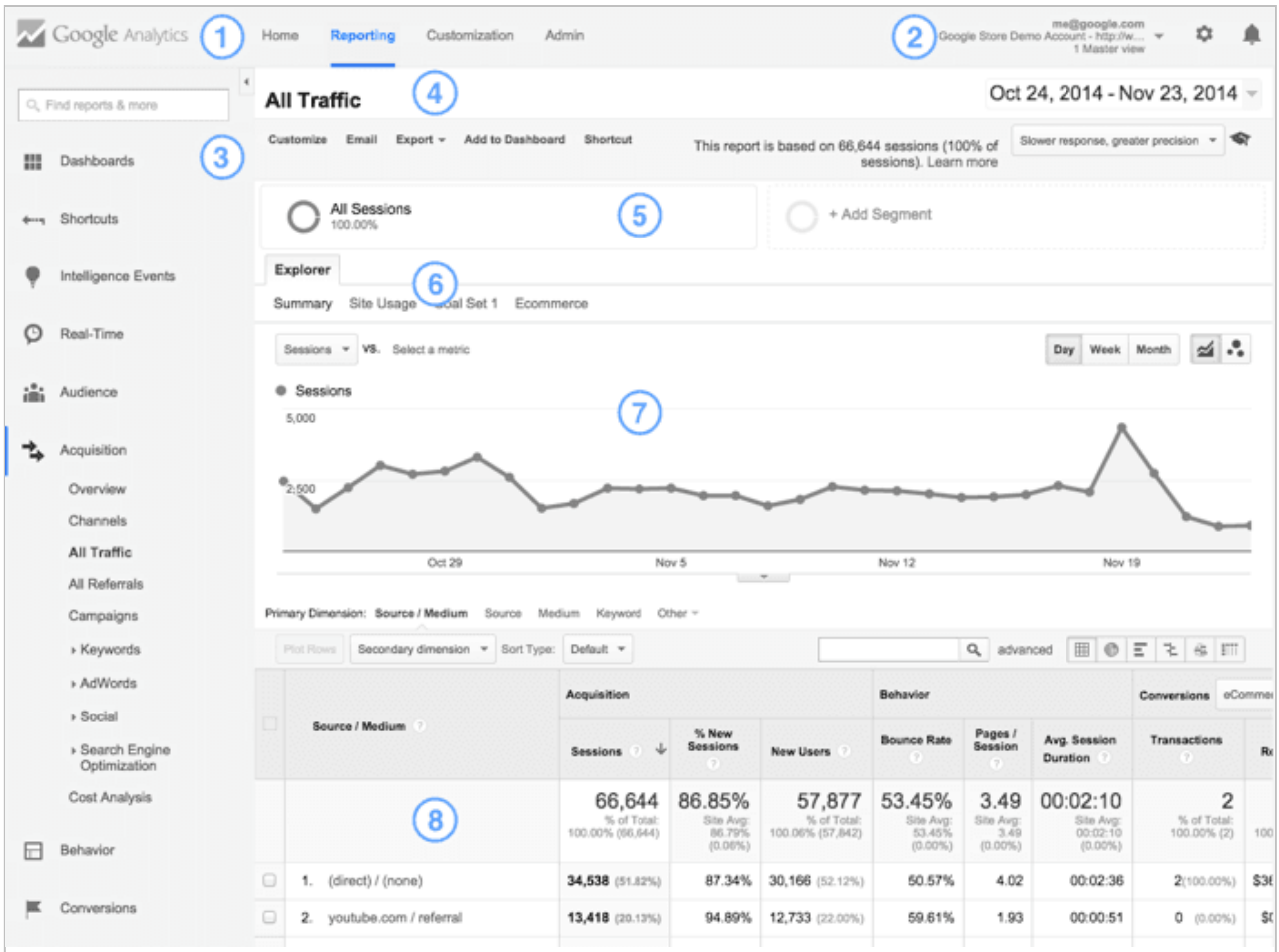
1. *Colour*
   - *many different colour models currently available*
   - *most useful relevant to dashboard design is the HSL model*
   - *this model describes colour in terms of three attributes*
     - *hue*
     - *saturation*
     - *lightness*
   - *perception of colour often depends upon context*
2. *Form*
   - *correct use of length, width, and general size can convey quantitative dimensions*
   - *each with varying degrees of precision*
   - *use the Laws of Prägnanz to manipulate groups of similar shapes and designs*
   - *thereby easily grouping like data and information for the user*
3. *Position*
   - *relative positioning of elements helps communicate dashboard information*
   - *laws of Prägnanz teach us*
   - *position can often infer a perception of relationship and similarity*
   - *higher items are often perceived as being better*
   - *items on the left of the screen traditionally seen first by a western user*

Data visualisation

Building a dashboard

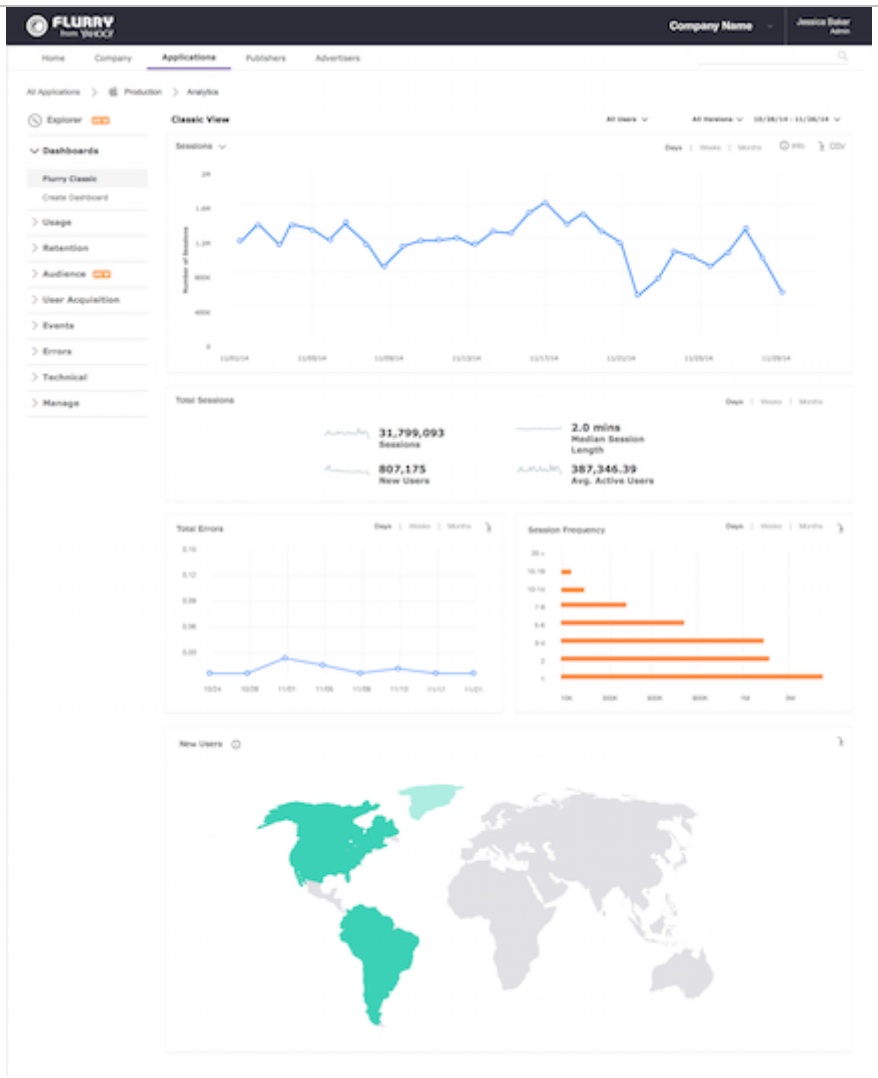
- need to clearly determine the questions that need to be answered
  - *given the information collated and presented within the dashboard*
- need to ensure that any problems can be detected on time
- be certain why we actually need a dashboard for the current dataset
- then begin to collect the requisite data to help us answer such questions
  - *data can be sourced from multiple, disparate datasets*
- chosen visualisations help us tell this story more effectively
- present it in a manner appealing to our users
- need to consider information visualisations familiar to our users
  - *helps reduce any potential user's cognitive overload*
- carefully consider organisation of data and information
- organise the data into logical units of information
  - *helps present dashboard information in a meaningful manner*
- dashboard sections should be organised
  - *to help highlight and detect any underlying or prevailing issues*
  - *then present them to the user*

Image - Google Analytics



Dashboard - Google Analytics

Image - Yahoo Flurry



Dashboard - Yahoo Flurry

Image - Mint

Dashboard - Mint

Data visualisation - D3

Intro - part 1

- D3 is a custom JavaScript library
  - *designed for the manipulation of data centric documents*
  - *uses a custom library with HTML, CSS, and SVG*
  - *creates graphically rich, informative documents for the presentation of data*
- D3 uses a data-driven approach to manipulate the DOM
- Setup and configuration of D3 is straightforward
  - *most involved aspect is the configuration of a web server*
- D3.js works with standard HTML files
  - *requires a web server capable of parsing and rendering HTML...*
- to parse D3 correctly we need
  - *UTF-8 encoding reference in a meta element in the head section of our file*
  - *reference D3 file, CDN in standard script element in HTML*

Data visualisation - D3

intro - part 2

- D3 Wiki describes the underlying functional concepts as follows,

> *D3's functional style allows code reuse through a diverse collection of components and plugins.*
>
> *D3 Wiki*

- in JS, functions are objects
  - *as with other objects, a function is a collection of a name and value pair*
- real difference between a function object and a regular object
  - *a function can be invoked, and associated, with two hidden properties*
  - *include a function context and function code*
- variable resolution in D3 relies on variable searching being performed locally first
- if a variable declaration is not found
  - *search will continue to the parent object*
  - *continue recursively to the next static parent*
  - *until it reaches global variable definition*
  - *if not found, a reference error will be generated for this variable*
- important to keep this static scoping rule in mind when dealing with D3

Data visualisation - D3

Data Intro - part 1

- Data is structured information with an inherent perceived potential for meaning
- consider data relative to D3
  - *need to know how data can be represented*
  - *both in programming constructs and its associated visual metaphor*
- what is the basic difference between data and information?

> *Data are raw facts. The word raw indicates that the facts have not yet been processed >>> to reveal their meaning...Information is the result of processing raw data to reveal >>> its meaning.*
>
> *Rob, Morris, and Coronel. 2009*

- a general concept of data and information
- consider them relative to visualisation, impart a richer interpretation
- information, in this context, is no longer
  - *the simple result of processed raw data or facts*
  - *it becomes a visual metaphor of the facts*
- same data set can generate any number of visualisations
  - *may lay equal claim in terms of its validity*
- visualisation is communicating creator's insight into data...

Data visualisation - D3

Data Intro - part 2

- relative to development for visualisation
  - *data will often be stored simply in a text or binary format*
- not simply textual data, can also include data representing
  - *images, audio, video, streams, archives, models...*
- for D3 this concept may often simply be restricted to
  - *textual data, or text-based data...*
  - *any data represented as a series of numbers and strings containing alpha numeric characters*
- suitable textual data for use with D3
  - *text stored as a comma-separated value file (.csv)*
  - *JSON document (.json)*
  - *plain text file (.txt)*
- data can then be *bound* to elements within the DOM of a page using D3
  - *inherent pattern for D3*

Data visualisation - D3

Data Intro - Enter-Update-Exit Pattern

- in D3, connection between data and its visual representation
  - *usually referred to as the* ***enter-update-exit*** *pattern*
- concept is starkly different from the standard imperative programming style
- pattern includes
  - *enter mode*
  - *update mode*
  - *exit mode*

Data visualisation - D3

Data Intro - Enter-Update-Exit Pattern

Enter mode

- `enter()` function returns all specified data that not yet represented in visual domain
- standard modifier function chained to a selection method
  - *create new visual elements representing given data elements*
  - *eg: keep updating an array, and outputting new data bound to elements*

Update mode

- `selection.data(data)` function on a given selection
  - *establishes connection between data domain and visual domain*
- returned result of intersection of data and visual will be a **data-bound** selection
- now invoke a modifier function on this newly created selection
  - *update all existing elements*
  - *this is what we mean by an **update** mode*

Exit mode

- invoke `selection.data(data).exit` function on a data-bound selection
  - *function computes new selection*
  - *contains all visual elements no longer associated with any valid data element*
- eg: create a bar chart with 25 data points
  - *then update it to 20, so we now have 5 left over*
  - ***exit mode** can now remove excess elements for 5 spare data points*

Data visualisation - D3

Data Intro - binding data - part 1

- consider standard patterns for working with data
- we can iterate through an array, and then bind the data to an element
  - *most common option in D3 is to use the* ***enter-update-exit*** *pattern*
- use same basic pattern for binding object literals as data
- to access our data we call the required attribute of the supplied data

```
var data = [
    {height: 10, width: 20},
    {height: 15, width: 25}
];


function (d) {
    return (d.width) + "px";
}
```

- then access the **height** attribute per object in the same manner
- we can also bind functions as data
  - *D3 allows functions to be treated as data...*

Data visualisation - D3

Data Intro - binding data - part 2

- D3 enables us to bind data to elements in the DOM
  - *associating data to specific elements*
  - *allows us to reference those values later*
  - *so that we can apply required mapping rules*
- use D3's `selection.data()` method to bind our data to DOM elements
  - *we obviously need some data to bind, and a selection of DOM elements*
- D3 is particularly flexible with data
  - *happily accepts various types*
- D3 also has a built-in function to handle loading JSON data

```
d3.json("testdata.json", function(json) {
    console.log(json); //do something with the json...
});
```

Data visualisation - D3

Data Intro - working with arrays - options

- min and max = return the min and max values in the passed array

```
d3.select("#output").text(d3.min(ourArray));
d3.select("#output").text(d3.max(ourArray));
```

- extent = retrieves both the smallest and largest values in the the passed array

```
d3.select("#output").text(d3.extent(ourArray));
```

- sum

```
d3.select("#output").text(d3.sum(ourArray));
```

- median

```
d3.select("#output").text(d3.median(ourArray));
```

- mean

```
d3.select("#output").text(d3.mean(ourArray));
```

- asc and desc

```
d3.select("#output").text(ourArray.sort(d3.ascending));
d3.select("#output").text(ourArray.sort(d3.descending));
```

- & many more...

Data visualisation - D3

Data Intro - working with arrays - nest

- D3's nest function used to build an algorithm
  - *transforms a flat array data structure into a hierarchical nested structure*
- function can be configured using the key function chained to **nest**
- nesting allows elements in an array to be grouped into a hierarchical tree structure
  - *similar in concept to the group by option in SQL*
  - ***nest** allows multiple levels of grouping*
  - *result is a tree rather than a flat table*
- levels in the tree are defined by the *key* function
- leaf nodes of the tree can be sorted by value
- internal nodes of the tree can be sorted by key

Data visualisation - D3

Selections - intro

- **Selection** is one of the key tasks required within D3 to manipulate and visualise our data
- simply allows us to target certain visual elements on a given page
- Selector support is now standardised upon the W3C specification for the Selector API
  - *supported by all of the modern web browsers*
  - *its limitations are particularly noticeable for work with visualising data*
- Selector API only provides support for selector and not selection
  - *able to select an element in the document*
  - *to manipulate or modify its data we need to implement a standard loop etc*
- D3 introduced its own selection API to address these issues and perceived shortcomings
  - *ability to select elements by ID or class, its attributes, set element IDs and class, and so on...*

Data visualisation - D3

Selections - single element

- select a single element within our page

```
d3.select("p");
```

- now select the first <p> element on the page, and then allow us to modify as necessary
  - *eg; we could simply add some text to this element*

```
d3.select("p")
.text("Hello World");
```

- selection could be a generic element, such as <p>
  - *or a specific element defined by targeting its ID*
- use additional modifier functions, such as `attr`, to perform a given modification on the selected element

```
//set an attribute for the selected element
d3.select("p").attr("foo");
//get the attribute for the selected element
d3.select("p").attr("foo");
```

- also add or remove classes on the selected element

```
//test selected element for specified class
d3.select("p").classed("foo")
//add a class to the selected element
d3.select("p").classed("goo", true);
//remove the specified class from the selected element
d3.select("p").classed("goo", function(){ return false; });
```

Data visualisation - D3

Selections - multiple elements

- also select all of the specified elements using D3

```
d3.selectAll("p")
.attr("class", "para");
```

- use and implement multiple element selection
  - *same as single selection pattern*
- also use the same modifier functions
- allows us to modify each element's attributes, style, class...

Data visualisation - D3

Selections - iterating through a selection

- D3 provides us with a selection iteration API
  - *allows us to iterate through each selection*
  - *then modify each selection relative to its position*
  - *very similar to the way we normally loop through data*

```
d3.selectAll("p")
.attr("class", "para")
.each(function (d, i) {
    d3.select(this).append("h1").text(i);
});
```

- D3 selections are essentially like arrays with some enhancements
  - *use the iterative nature of Selection API*

```
d3.selectAll('p')
.attr("class", "para2")
.text(function(d, i) {
    return i;
});
```

Data visualisation - D3

Selections - performing sub-selection

- for selections - often necessary to perform specific scope requests
  - *eg: selecting all <p> elements for a given element*

```
//direct css selector (selector level-3 combinators)
d3.select("div > p")
    .attr("class", "para");

//d3 style scope selection
d3.select("div")
    .selectAll("p")
    .attr("class", "para");
```

- both examples produce the same effect and output, but use very different selection techniques
  - *first example uses the CSS3, level-3, selectors*
  - *`div > p` is known as combinators in CSS syntax*

Data visualisation - D3

Selections - combinators

Example combinators..

1. descendant combinator

- uses the pattern of `selector selector` - describing loose parent-child relationship
- loose due to possible relationships - parent-child, parent-grandchild...

```
d3.select("div p");
```

- select the <p> element as a child of the parent `<div>` element
  - *relationship can be generational*

2. child combinator

- uses same style of syntax, `selector > selector`
- able to describe a more restrictive **parent-child** relationship between two elements

```
d3.select("div > p");
```

- finds <p> element if it is a direct child to the `<div>` element

Data visualisation - D3

Selections - D3 sub-selection

- sub-selection using D3's built-in selection of child elements
- a simple option to select an element, then chain another selection to get the child element
- this type of chained selection defines a scoped selection within D3
  - *eg: selecting a <p> element nested within our selected element*
  - *each selection is, effectively, independent*
- D3 API built around the inherent concept of function chaining
  - *can almost be considered a Domain Specific Language for dynamically building HTML/SVG elements*
- a benefit of chaining = easy to produce concise, readable code

```
var body = d3.select("body");


body.append("div")
    .attr("id", "div1")
  .append("p")
    .attr("class", "para")
  .append("h5")
    .text("this is a paragraph heading...");
```

Data visualisation - D3

Data Intro - page elements

- generation of new DOM elements normally fits
  - *either circles, rectangles, or some other visual form that represents the data*
- D3 can also create generic structural elements in HTML, such as a <p>
  - *eg: we can append a standard p element to our new page*

```
d3.select("body").append("p").text("sample text...");
```

- used D3 to select `body` element, then append a new <p> element with text "new paragraph"
- D3 supports *chain syntax*
  - *allowed us to `select`, `append`, and add `text` in one statement*

Data visualisation - D3

Data Intro - page elements

```
d3.select("body").append("p").text("sample text...");
```

- `d3`
  - *references the D3 object, access its built-in methods*
- `.select("body")`
  - *accepts a CSS selector, returns first instance of the matched selector in the document's DOM*
  - *`.selectAll()`*
  - ***NB:** this method is a variant of the single `select()`*
  - *returns all of the matched CSS selectors in the DOM*
- `.append("p")`
  - *creates specified new DOM element*
  - *appends it to the end of the defined select CSS selector*
- `.text("new paragraph")`
  - *takes defined string, "new paragraph"*
  - *adds it to the newly created <p> DOM element*

Data visualisation - D3

Binding data - making a selection

- choose a selector within our document
  - *eg: we could select all of the paragraphs in our document*

```
d3.select("body").selectAll("p");
```

- if the element we require does not yet exist
  - *need to use the method `enter()`*

```
d3.select("body").selectAll("p").data(dataset).enter().append("p").text("new paragraph");
```

- we get new paragraphs that match total number of values currently available in the **dataset**
  - *akin to looping through an array*
  - *outputting a new paragraph for each value in the array*
- create new, data-bound elements using `enter()`
  - *method checks the current DOM selection, and the data being assigned to it*
- if more data values than matching DOM elements
  - *`enter()` creates a new placeholder element for the data value*
  - *then passes this placeholder on to the next step in the chain, eg: `append()`*
- data from dataset also assigned to new paragraphs
- **NB:** when D3 binds data to a DOM element, it does not exist in the DOM itself
  - *it does exist in the memory*

Data visualisation - D3

Binding data - using the data

- change our last code example as follows,

```
d3.select("body").selectAll("p").data(dataset).enter().append("p").text(function(d) { return d; });
```

- then load our HTML, we'll now see dataset values output instead of fixed text
- anytime in the chain after calling the `data()` method
  - *we can then access the current data using d*
- also bind other things to elements with D3, eg: CSS selectors, styles...

```
.style("color", "blue");
```

- chain the above to the end of our existing code
  - *now bind an additional css style attribute to each <p> element*
  - *turning the font colour blue*
- extend code to include a conditional statement that checks the value of the data
  - *eg: simplistic striped colour option*

```
.style("color", function(d) {
if (d % 2 == 0) {
return "green";
} else {
 return "blue";
}
});
```

- DEMO - D3 basic elements

Image - D3 Basic Elements

D3 - basic elements

Source code - demos

D3.js

- D3 basic elements

MongoDB

- 424-node-mongo1

References

- Chocolatey for Windows
  - *Chocolatey package manager for Windows*
- D3.js
  - *D3 - API reference*
  - *D3 - Wiki*
- Homebrew for OS X
  - *Homebrew - the missing package manager for OS X*
- Kirk, A. *Data Visualisation: A successful design process*. Packt Publishing. 2012.
- MongoDB
  - *MongoDB - For Giant Ideas*
  - *MongoDB - Getting Started (Node.js driver edition)*
  - *MongoDB - Getting Started (shell edition)*
- Mongoose
  - *MongooseJS Docs*
- Node.js
  - *Node.js home*
  - *Node.js - download*
- W3 Selector API