Comp 388/424 - Client-side Web Design

Spring Semester 2016 - Week 12

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 - intro

- MongoDB is another example of a NoSQL based data store
  - *a database that enables us to store our data on disk*

- unlike MySQL, for example, it is not in a relational format
- MongoDB is best characterised as a **document-oriented** database
- conceptually may be considered as storing objects in collections
- stores its data using the BSON format
  - *consider similar to JSON*
  - *use JavaScript for working with MongoDB*

Server-side considerations - data storage

MongoDB - document oriented

- SQL database, data is stored in tables and rows

- MongoDB, by contrast, uses **collections** and **documents**

- comparison often made between a collection and a table

- **NB:** a document is quite different from a table or row...

- a document may contain a lot more data than a table or row...

- a noted concern with this document approach is duplication of data

- one of the trade-offs between NoSQL (MongoDB) and SQL

- SQL - goal of data structuring is to normalise as much as possible

- thereby avoiding duplicated information

- NoSQL (MongoDB) - provision a data store, as easy as possible for the application to use

Server-side considerations - data storage

MongoDB - BSON

- BSON is the format used by MongoDB to store its data

- effectively, JSON stored as binary with a few notable differences
  - *eg: `ObjectId` values - data type used in MongoDB to uniquely identify documents*
  - *created automatically on each document in the database*
  - *often considered as analogous to a primary key in a SQL database*

- `ObjectId` is a large pseudo-random number

- for nearly all practical occurrences, assume number will be unique

- might cease to be unique if server can't keep pace with number generation...

- other interesting aspect of `ObjectId`
  - *they are partially based on a timestamp*
  - *helps us determine when they were created*

Server-side considerations - data storage

MongoDB - general hierarchy of data

- in general, MongoDB has a three tiered data hierarchy

1. database
  - *normally one database per app*
  - *possible to have multiple per server*
  - *same basic role as DB in SQL*

2. collection
  - *a grouping of similar pieces of data*
  - *documents in a collection*
  - *name is usually a noun*
  - *resembles in concept a table in SQL*
  - *documents do not require the same schema*

3. document
  - *a single item in the database*
  - *data structure of field and value pairs*
  - *similar to objects in JSON*
  - *eg: an individual user record*

Server-side considerations - data storage

MongoDB - install and setup

- install on Linux

- install on Mac OS X
  - *again, we can use **Homebrew** to install MongoDB*

```
// update brew packages
brew update
// install MongoDB
brew install mongodb
```

- then follow the above OS X install instructions to set paths...

- install on Windows

Server-side considerations - data storage

MongoDB - a few shell commands

- issue following commands at command line to get started - OS X etc

```
// start MongoDB server - terminal window 1
mongod
// connect to MongoDB - terminal window 2
mongo
```

- switch to, create a new DB (if not available), and drop a current DB as follows

```
// list available databases
show dbs
// switch to specified db
use 424db1
// show current database
db
// drop current database
db.dropDatabase();
```

- DB is not created permanently until data is created and saved
  - *insert a record and save to current DB*

- only permanent DB is the local `test` DB, until new DBs created...

Server-side considerations - data storage

MongoDB - a few shell commands

- add an initial record to a new `424db1` database.

```
// select/create db
use 424db1
// insert data to collection in current db
db.notes.insert({
...     "travelNotes": [{
...     "created": "2015-10-12T00:00:00Z",
...     "note": "Curral das Freiras..."
...     }]
... })
```

- our new DB `424db1` will now be saved in Mongo

- we've created a new collection, `notes`

```
// show databases
show dbs
// show collections
show collections
```

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*

```javascript
...
//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
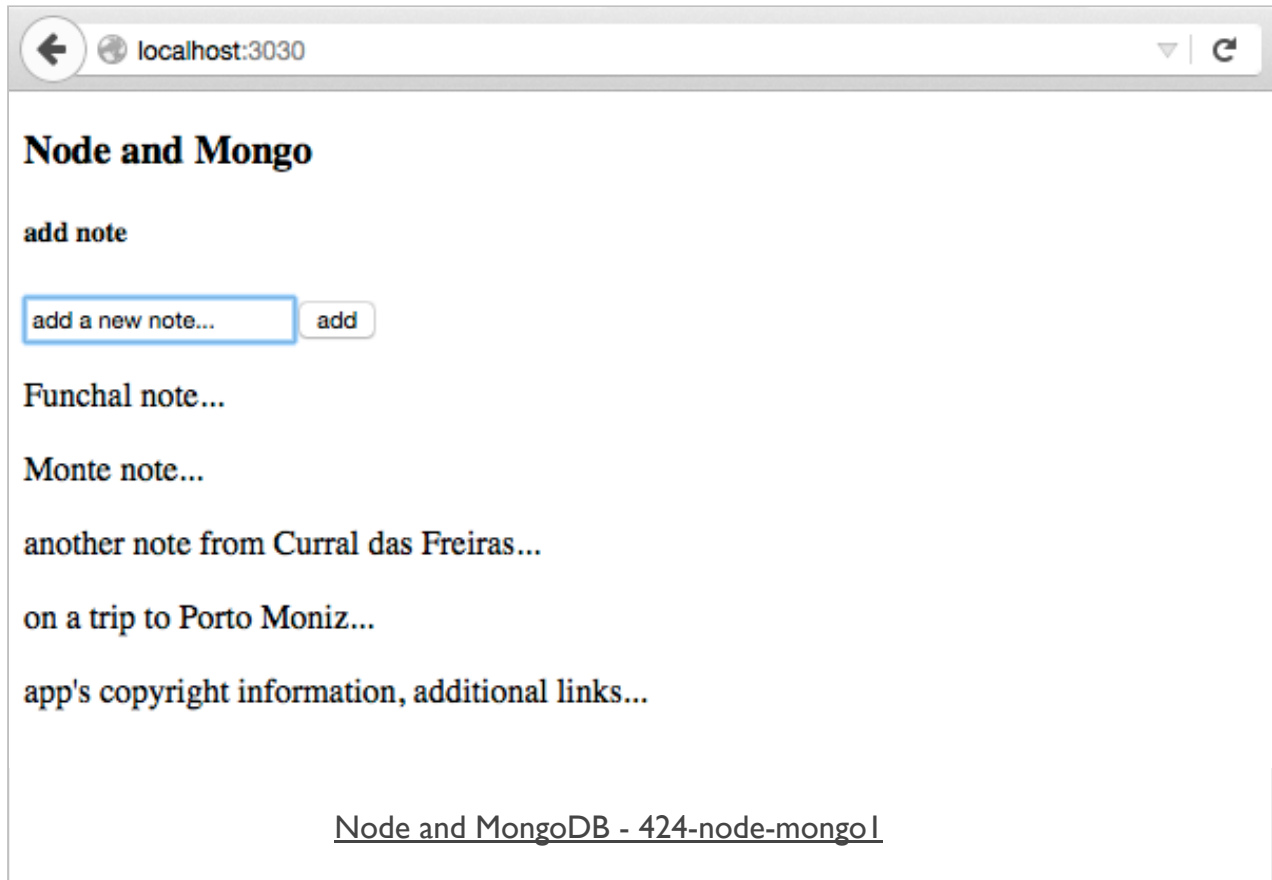
Data visualisation

- 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

- 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

- 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

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

- 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

- 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

- 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*
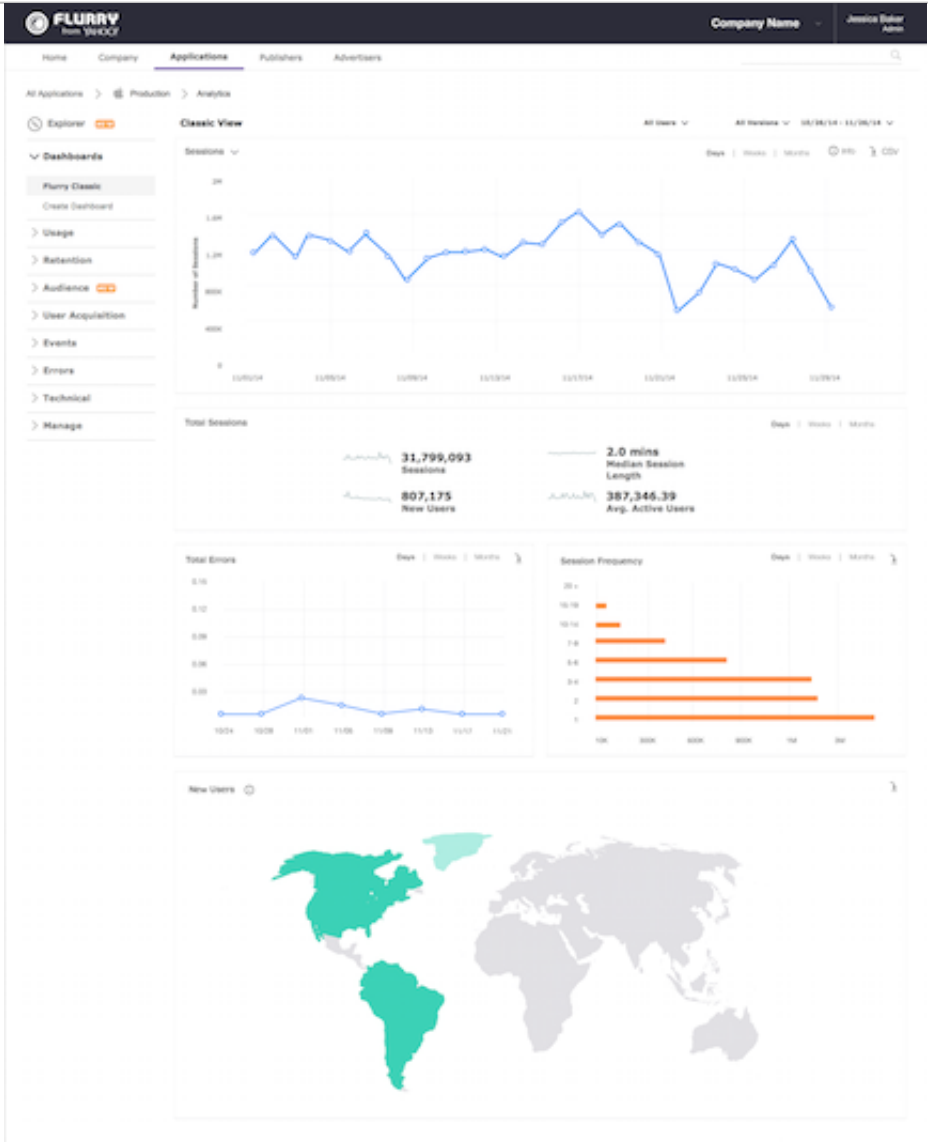
Data visualisation

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
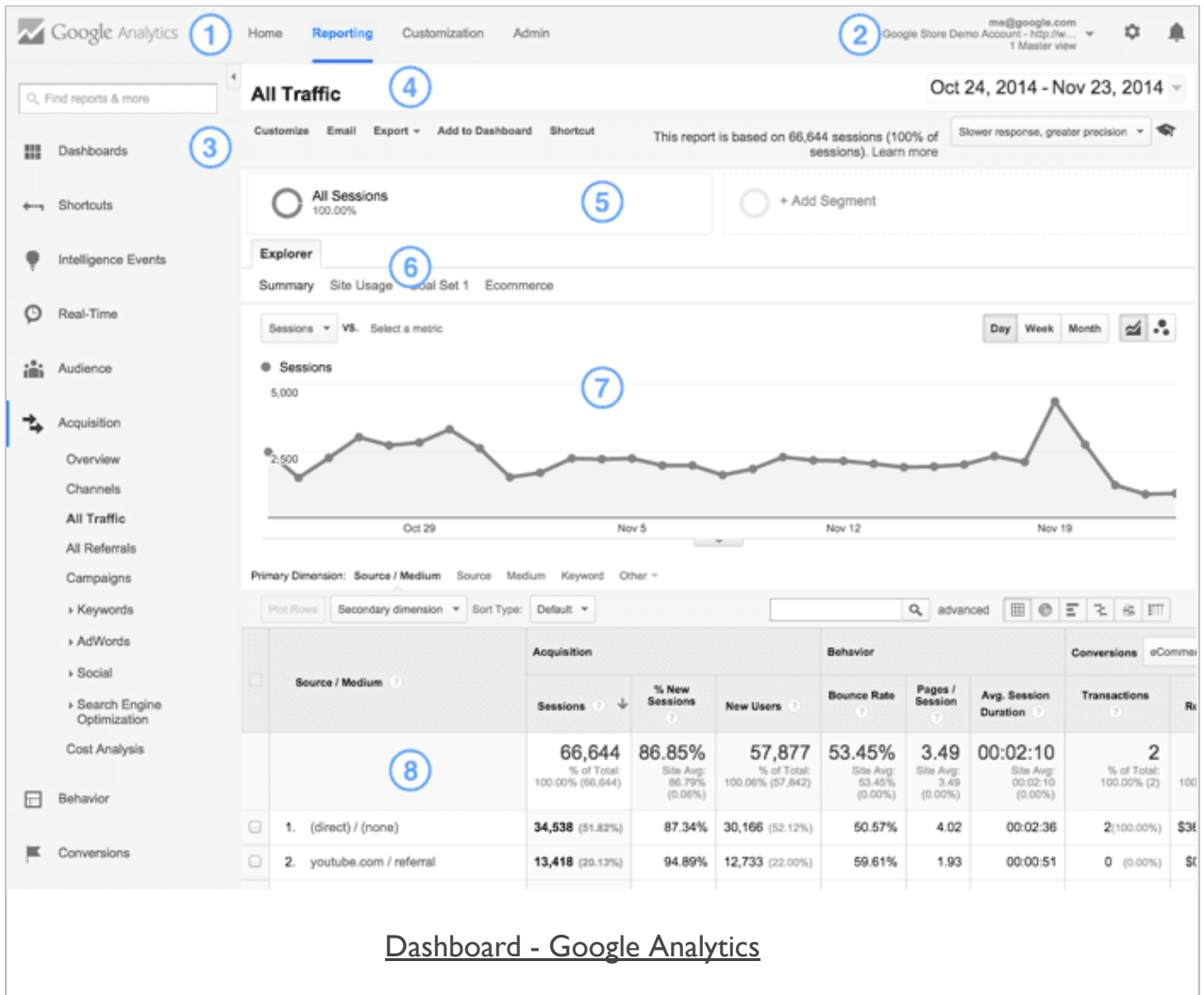
Dashboard - Google Analytics

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

- 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 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

- 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 I

- 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

```javascript
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

- **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...*

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