Comp 336/436 - Markup Languages

Fall Semester 2019 - Week 13

Dr Nick Hayward

Final assessment

Course total = 40%

- working final app
- modify and update DEV week project work
  - *include parsing and rendering of the data suitable for broader publication*
  - *develop and integrate stronger semantic organisation and manipulation*
  - *provide an opportunity for analysis, visualisation...*
- presentation and demo - live working app
  - *due on Wednesday 4th December 2019 @ 4.15pm*
  - *show and explain implemented differences from DEV week project*
  - *where and why did you update the app?*
  - *benefits of updates?*
- how did you respond to the feedback review?
- final report
  - *due on Wednesday 11th December 2019 @ 4.15pm*

XML & Semantic Web

- semantic web as a broad grouping of organised data
- different XML-based recommendations
- RDF (Resource Description Framework)
  - *XML based text format*
  - *supports resource description and metadata apps*
- GRDDL (Gleaning Resource Descriptions from Dialects of Languages)
  - *introduces markup based on existing standards*
  - *declare XML document with data compatible with RDF*
  - *add linking to algorithms - usually as XSLT*
  - *used to extract data from document...*
- OWL (Web Ontology Language)
  - *semantic markup language for publishing and sharing ontologies*
  - *OWL is a vocabulary extension of RDF*
  - *derived from DAML+OIL Web Ontology Language*
- SPARQL
  - *query language for RDF*
  - *used to express queries across diverse data sources*

Metadata typology and uses - intro

> *the main goal of metadata, whether considered as a theory or implemented in a functional system, is to make data useful...*

- Garoufallou, E., Greenberg, J., *Metadata and Semantics Research: 7th International Conference*. MSTR. 2013.
- scholars and practitioners need to ascribe types to things
- classify kinds of *things*
  - *enables better understanding of variety and complexity...*
- need to typify metadata
  - *also metadata schemas, metadata uses, metadata elements, metadata values...*
- **types of metadata** includes different concepts
  - *types of metadata standards*
  - *types of metadata values*
  - *types of metadata functions*
  - *and types of metadata elements*

Metadata typology and uses - which types?

- common metadata usage is *Dublin Core*
  - *often perceived as a catch-all solution...*
- many different options available for metadata, e.g.
  - *metadata for preservation*
  - *metadata for intellectual property rights*
  - *METS*
  - *metadata for geospatial information*
  - *AACR*
  - *metadata for describing music*
  - *FRBR, LOM, RDA, RDF...*
- often use a mixture of models, domains, uses...
  - *disparate parts forming our understanding of metadata*

Metadata typology and uses - typify

- need *criteria* to typify something
  - *and potentially everything in a given domain...*
- simple term *metadata* and simple definition, *data about data*
  - *encompass a complex technical and intellectual infrastructure*
  - *used to manage and retrieve digital objects in different digital contexts*
  - *within different digital information systems and services*
- criteria often matches sheer diversity of digital information management systems
- could be applied to different metadata levels, e.g.
  - *records, agents, elements, schemas...*

Metadata typology and uses - classify metadata

- metadata may describe different attributes or properties of information objects
  - *giving them meaning, context and organisation in a standardised way*
- many uses and dimensions of metadata
  - *has now led to construction of a very broad typology*
- use obvious criteria to typify metadata
- might classify metadata, mixing different levels, e.g.
  - *systems, practices, models, schemas, elements, records and trends*
- mix different levels with seven criteria, e.g.
  - *criterion 1: way of creation...*
  - *criterion 2: moment of creation...*
  - *criterion 3: way of storage...*
  - *criterion 4: level of structure...*
  - *criterion 5: purpose of the metadata...*
    - domain independent metadata
    - domain dependent metadata
  - *criterion 6: application...*
    - i.e. what's metadata used for?
  - *criterion 7: level of standardisation*

Metadata typology and uses - types and systems

- often consider metadata types to help understand associated metadata systems
- need to distinguish different types of metadata
  - *to help categorise metadata*
- classification can mix different levels to categorise metadata, e.g.
  - *record level, element level or schema/element set level*
- we often consider the following
  - *general versus specialist*
  - *minimalist versus rich*
  - *hierarchical versus linear*
  - *structured versus unstructured*
  - *machine generated versus human authored*
  - *embedded versus detached*
- **surface** metadata
  - *information that can be gathered by machines and converted into metadata*
  - *process of gathering it is often known as* ***screen scraping***
- *other types of metadata*
  - *including keywords, Google, tags, user created metadata, &c.*
  - *self-description mechanisms for social tagging and new trends*
  - *avoid keyword stacking*
  - *Kartus, E. 2006.*

Metadata typology and uses - types and vocabularies

- purpose of any metadata, whether element or model
  - *describe a property or set of useful properties*
  - *for a given information resource or object*
- many current metadata systems include an inherent multidimensional nature
  - *such structures often called metadata vocabularies*
- a few useful definitions
  - *metadata models/formats help express properties of a resource, e.g. a subject*
  - *metadata schemas = metadata formats encoded in a standardised machine readable markup language*
  - *metadata scheme = a set of rules or terms for encoding the value of a particular metadata term*
  - *both schemas and schemes involve metadata elements*

Metadata typology and uses - formats and schemas

- in the 1990's, common to speak about *metadata formats* or *metadata models*
  - *to refer to a set of properties*
  - *expressed and defined in a standardised way*
  - *served to describe a digital information object*
- each description constituted a metadata record
  - *applying that metadata format to a particular object*
- grouped metadata formats along a continuum of growing structural and semantic richness
  - *used to identify shared characteristics of each grouping*
  - *patterns from paths of the given metadata formats*
  - *Dempsey, L. & Heery, R. 1998.*

Metadata typology and uses - schema/scheme

- DCMI Glossary definition for *schema/scheme*

> *any organization, coding, outline or plan of concepts. In terms of metadata, a systematic, orderly combination of elements or terms.*

- when a declaration of metadata terms is represented in XML or RDF schema language
  - *it might be better considered strictly a schema*
- schemas are machine-processable specifications
  - *define structure and syntax of metadata specifications in a formal way*
- difficult to think of a single metadata element or a single metadata value
  - *in isolation, without broader context*
- current Semantic Web approach
  - *considers schemas and schemes*
  - *assumes they are formalised in a machine-understandable way*
- *schemas* are sets of metadata elements and rules
  - *previously stated for a particular purpose*
- *schemes* are a set of rules for encoding information
  - *supports a specific information domain*
- schemas and schemes are commonly expressed in XML/RDF vocabularies
  - *easier to read and use*
- also form a perceived complete metadata infrastructure
  - *for current digital information services*
  - *becoming Metadata Vocabularies*
- metadata classification starting from metadata vocabularies

- *metadata schemas, metadata schemes, and metadata elements*

Metadata typology and uses - types of schemas

- current metadata schemas include
  - *not only the semantics of an entire element set*
  - *but also encoding of elements and structure with a markup language*
- two crucial criteria to take into account
  - *to categorise the complexity and variety of current metadata*
  - ***granularity*** *and the* ***information domain***

Metadata typology and uses - types of schemas - granularity

- draw up a typology of metadata schemas
  - *based on hierarchical relations between different sets of metadata*
- extensible nature of metadata, e.g.
  - *global metadata schemas:*
    - e.g. Dublin Core, which is the best known example
  - *local metadata schemas:*
    - e.g. SEPIADES, Europeana Semantic Elements...
  - *container metadata schemas:*
    - container architectures
    - e.g. RDF, METS, ONIX, MARC, &c.
  - *conceptual metadata schemas*
    - e.g. CIDOC-CRM

Metadata typology and uses - types of schemas - domain

- an example of domain specific is *Web-based information*
  - *sectioned in a vertical way*
- there are global search engines, and very large digital libraries like Europeana
- increasingly, there are also different portals
  - *digital libraries*
  - *and other digital information systems and services*
- locations where the objects you can find belong to a category
  - *either subject-oriented or type-oriented*
- a few domain specific examples
  - *metadata for Cultural Heritage*
  - *metadata for geographic and geospatial information systems*
  - *metadata for educational information systems*
  - *metadata for digital preservation/curation*

Metadata typology and uses - types of schemas - domain examples

metadata for Cultural Heritage

- often applied to cultural objects and visual resources
- in this huge information context, many early developed metadata schemas
- schemas range from those influenced by the librarian's domain
  - *to those built from archives and museums*
  - *or arts and architecture domains...*
- for example,
  - *traditional cataloging standards converted into schemas - MARCXML, MODS...*
  - *metadata standards for Finding Aids - Encoding Archival Description (EAD)*
  - *representation of digital texts - Text Encoding Initiative (TEI)*
  - *metadata schemas/standards for digital visual arts - CDWA...*

Metadata typology and uses - types of schemas - domain examples

metadata for geographic and geospatial information systems

- metadata in this particular domain has a clear protagonist, and almost standard
- *Content Standard for Digital Geospatial Metadata* (CSDGM)
  - *originally adopted for the Federal Geographic Data Committee in 1994*
  - *and then revised in 1998*
- its main elements were embraced by the international community
- adopted through the ISO metadata standard *ISO 19115*

Metadata typology and uses - types of schemas - domain examples

metadata for educational information systems

- another big domain to analyse specific metadata schemas and standards
  - *the educational and learning environment*
- *Learning Objects Metadata* (LOM)
  - *developed by the IEEE Learning Technology Standards Committee*
  - *the main metadata schema in this field*
- learning objects used and reused for educational purposes
  - *a particular type of digital information object*
  - *requiring particular types of metadata schemas*
  - *helps describe their education-specific properties*
- *Learning Objects Metadata* has a more complicated structure than many other schemas
  - *integrates different kinds of metadata element*
  - *e.g. descriptive, administrative, and technical elements*

Metadata typology and uses - types of schemas - domain examples

metadata for digital preservation/curation

- digital preservation and curation is a broad domain and concept
- implies much more than metadata
  - *a broad range of systems and infrastructures*
  - *designed to preserve the usable life of digital objects*
- preservation metadata is a domain
  - *supports all processes associated with digital preservation*
- primary examples for preservation metadata as a specific domain
  - *Open Archival Information Systems (OAIS (ISO 14721)*
  - *PREMIS (Preservation Metadata: Implementation Strategies)*

Metadata typology and uses - types of schemas - domain

- domain approach to digital information
  - *increasingly manifest throughout digital information services*
- besides big metadata schemas based upon a specific domain
  - *there are many application specific profiles*
- these are data elements drawn from other metadata schemas
  - *then combined together*
  - *and used for more specific or even local application domains*
  - *e.g. for public sector information*
- also find application profiles based on one schema
  - *but tailored for a particular information community*
  - *e.g. Dublin Core Education Application Profile*

Metadata typology and uses - types of schemes

- schemes describe a particular way to encode *metainformation* describing a given resource
  - *similar to Anglo American Cataloging Rules (AACR), Cataloging Cultural Objects (CCO)...*
- schemes are also sets of terms or vocabularies
  - *only concern the possible values for a particular metadata element*
- *metadata schemes* are a range of values
  - *values that might be provided for an assertion about a resource*
  - *e.g. Date-time formats, authority lists, controlled vocabularies...*

Metadata typology and uses - types of schemes - examples

- *authority list* is a metadata scheme
  - *could be applied to encode the values of a metadata element*
  - *or term in a metadata schema*
- e.g. if we're dealing with a resource's authorship
  - *the element `DC.Creator` in Dublin Core*
  - *or `Author` in TEI Header*
  - …
- large-scale metadata content standards such as *Cataloging Cultural Objects* (CCO)
  - *include general instructions applicable to a metadata model or a set of its elements*
- metadata schemes are specific vocabularies
  - *devoted to the values of a particular metadata element*
- encoding schemes provide contextual information
  - *or parsing rules that aid in the interpretation of an element value*
- such contextual information may take different forms, e.g.
  - *controlled vocabularies, formal notations, or parsing rules*
- schemes might also be called *value spaces*
  - *forming the set of possible values for a given type of data*

Metadata typology and uses - types of schemes - domain

- as with schemas, schemes may be classified as well
- *general purpose schemes*
  - *e.g. universal classifications or generic subject heading lists...*
- *specific purpose schemes*
  - *e.g. traditional metadata schemes*
  - *thesauri, other vocabularies applied to a specific information domain...*
- many, many vocabularies encoded as metadata schemes...
  - *e.g. metadata vocabularies for cultural heritage*
  - *Union List of Artist Names (ULAN)*
  - *Arts and Architecture Thesaurus (AAT)*
  - ...
- schemes could be
  - *traditional thesauri*
  - *classification schemes*
  - *and other knowledge organisation systems...*
- schemes could also be
  - *subject-based metadata like ontologies and folksonomies*
  - *traditional vocabularies encoded for the Semantic Web*
  - *e.g. following a formal schema like Simple Knowledge Organization System (SKOS)*

Metadata typology and uses - types of schemes - ontologies

- most of the ontologies we find, for example, are also domain oriented
- they divide the realm of knowledge that they represent into:
  - *individuals*
  - *classes*
  - *attributes*
  - *relations*
  - *and events*

Metadata typology and uses - types of schemes - folksonomies

- semantically weaker *folksonomies* does not usually have a domain orientation
  - *they do not have any kind of control*
- a *folksonomy* is a record with labels, tags or keywords
  - *used by many people on the Web*
  - *usually without a particular purpose or initial structure*
- a *folksonony* will be more general
  - *dynamic in its growth and application*
- some experiences using social tagging within a specific domain
  - *e.g. Steve Museum - a systematic research project*
  - *considers how social tagging can best serve the museum community and its visitors*
  - *Steve Museum Project*
- in a distributed networked environment - rapid scheme changes
  - *managing semantics of changes vital to functioning and utility of schemes*

Metadata typology and uses - types of metadata - part 1

- many traditional classifications of metadata types are based on metadata elements
  - *assuming that every metadata schema has elements of similar types*
- allows fair comparison with known points of comparison
- almost every metadata handbook distinguish the following types of metadata
  - ***Descriptive*** *metadata elements*
  - ***Structural*** *metadata elements*
  - ***Administrative*** *metadata elements*
- types and functions of metadata elements classified into broad categories
  - *descriptive, structural, and administrative*
  - *they do not have well-defined boundaries and often overlap*
- different types blend into one another when using a specific metadata schema

Metadata typology and uses - types of metdata - part 2

- in general, metadata schemas and standards include these types of metadata elements
- e.g., **METS** packages structural, descriptive, administrative, and other metadata
  - *with an information object or digital surrogate*
  - *indicates types of relationships*
  - *e.g amongst different parts of the current complex information object*
- so, any metadata schema can classify their elements as different types of metadata
- a traditional categorisation using this pattern
  - *Dublin Core elements division*
  - *distinguishes three groups for its metadata elements*
- *Dublin Core* groupings
  - *Content elements - Title, Subject, Description, Source, Language, Relation, and Coverage*
  - *Intellectual Property elements - Creator, Publisher, Contributor, Rights*
  - *Instantiation elements - Date, Type, Format, and Identifier*

Metadata typology and uses - shaping standards

- several factors
  - *such as the type of resource and the application domain*
- help shape and influence the formation of metadata standards
- as digital information systems and services are increasingly vertical
  - *e.g. subject-oriented or community-oriented*
  - *metadata matches this trend as well...*
- identifying and monitoring actual use of metadata - not a straightforward process

Metadata typology and uses - user needs - part I

> *Setting metadata loose, through the internet and their widening user base, has in some cases resulted in new user contexts for existing metadata. The biodiversity domain provides an excellent illustration of how metadata which previously might have had a strong local focus and user community can attract a wider interest on the internet. International collaborative efforts, such as the European Network for Biodiversity Information (ENBI), are currently building large metadata repositories by aggregating metadata of local institutions through portals. Existing metadata are thus repurposed in the context of the international research on global warming and its impact on biodiversity.*

- Dempsey, L and R Heery. *Metadata: A current view of practice and issues*. Journal of Documentation, 54(2). PP.145–172. 1998.

Metadata typology and uses - user needs - part 2

- user needs regarding cultural heritage usually defined in general and vague terms
- museums, libraries, and archives have struggled to identify user needs
  - *lots of digitised material to access, not sure how it's being accessed...*
- user needs have attracted attention on a research level and within individual projects
- application of research outcomes and recommendations in the field remains problematic
- metadata practitioners rarely have chance to start from scratch
  - *issues with reviews, surveys, legacy requirements and standards...*
- metadata services rarely built from scratch...

Metadata typology and uses - use case evaluation

- no clear, standard set of practices or methodologies to monitor the use of metadata
- might monitor metadata using more traditional and resource intensive methods
  - *such as user surveys and interviews*
- there are a couple of examples for better monitoring usage of metadata
  - *dynamic search interfaces - last.fm, Facebook...*
  - *use case evaluation - collection management...*

Metadata typology and uses - use case evaluation - example

monitoring metadata usage with dynamic search interfaces

- marketers and HCI specialists often deduce user information
  - *regarding preferences of users*
  - *by analysing their behavior and actions*
  - *analyse through logfiles and tools such as heatmaps*
- *Facebook* and *last.fm* rely heavily on the analysis of use data
  - *helps deliver a more personalised service*
- *last.fm* tracks, for example, all of the music files played by the user
  - *allows them to fine-tune the user's profile*
  - *also allows them to offer extra features*
  - *features such as concert notification*
- *Facebook* is able to offer customised features, similar to *last.fm*

Metadata typology and uses - use case evaluation

- metadata fields are effectively used within a collection management database
- crucial to offer direct added value to users - encourages interaction with an application
- user is confronted with a default search interface
  - *may be intuitively configured to a user's individual needs*
  - *e.g. by adding, deleting, re-arranging metadata fields/elements...*
- possibility to monitor metadata fields usage within the database by a collections manager
- use recorded outcomes to manage relevance and usage of metadata fields
- provides statistical support for modifications and updates to metadata
- metadata providers need to start experimenting with tools and methodologies
  - *to allow them to monitor the effective use of the metadata*

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

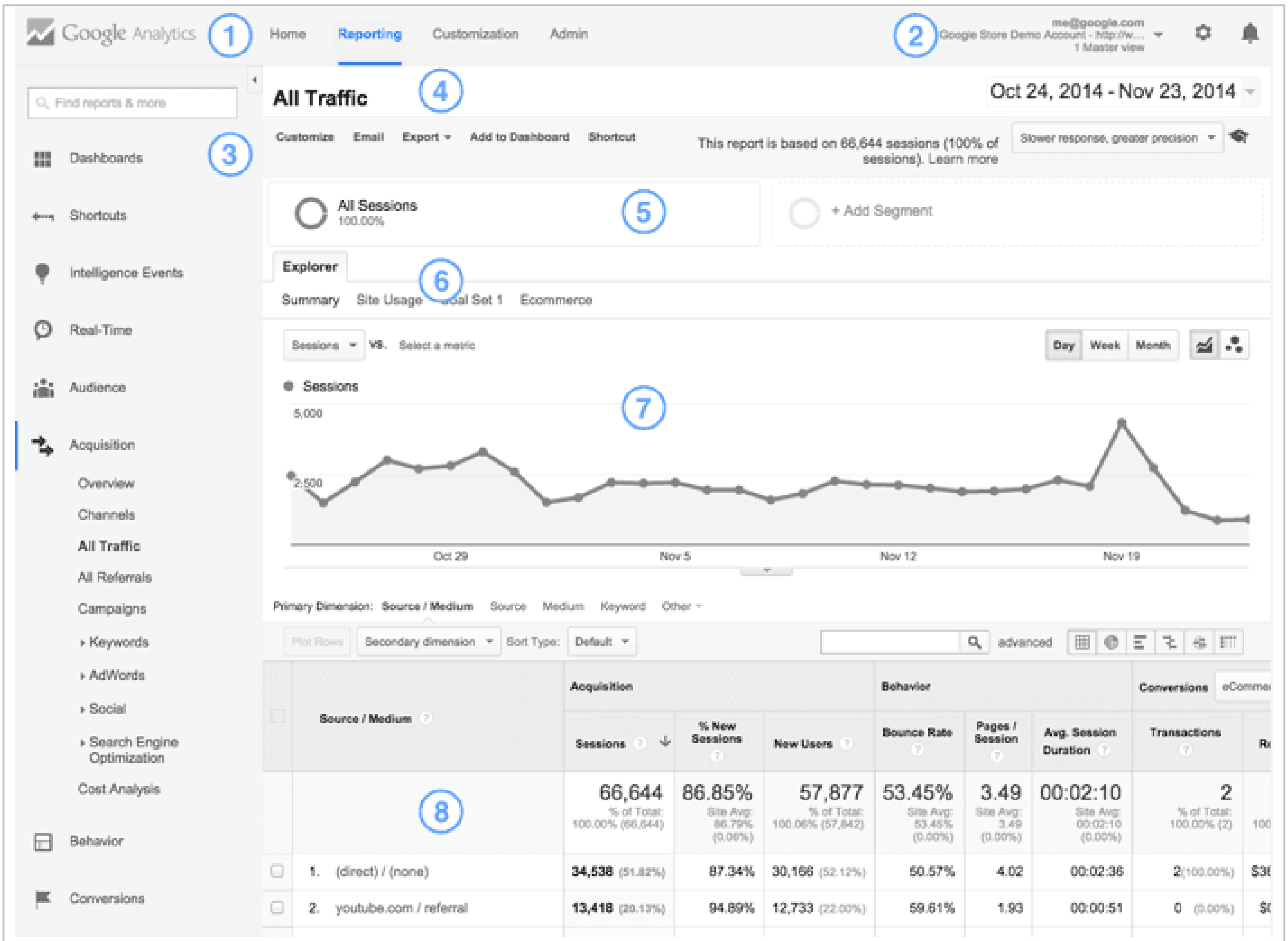
1. *Colour*
    - *many different colour models currently available*
    - *most useful relevant to dashboard design is the HSL model*
    - *this model describes colour in terms of three attributes*
        - *hue*
        - *saturation*
        - *lightness*
    - *perception of colour often depends 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

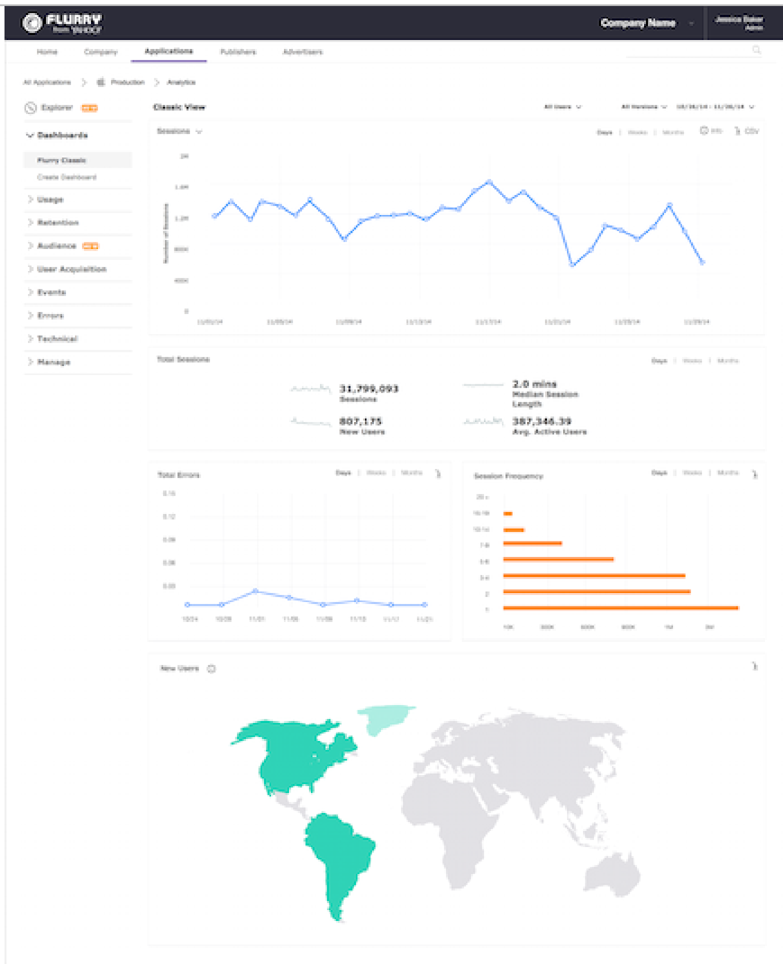
- need to clearly determine the questions that need to be answered
  - *given the information collated and presented within the dashboard*
- need to ensure that any problems can be detected on time
- be certain why we actually need a dashboard for the current dataset
- then begin to collect the requisite 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;
});
```

Selections - performing sub-selection

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

- 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(
```

- 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

Data visualisation - D3

Drawing - intro - part 1

1. drawing divs

- one of the easiest ways to draw a rectangle, for example, is with a HTML `<div>`
- an easy way to start drawing a bar chart for our stats
- start with standard HTML elements, then consider more powerful option of drawing with SVG
- semantically incorrect, we could use `<div>` to output bars for a bar chart
  - *use of an empty `<div>` for purely visual effect*
- using D3, add a class to an empty element using `selection.attr()` method

2. setting attributes

- `attr()` is used to set an HTML attribute and its value on an element
- After selecting the required element in the DOM
  - *assign an attributes as follows*

```
.attr("class", "barchart")
```

Data visualisation - D3

Drawing - intro - part 2

- use D3 to draw a set of bars in divs as follows

```
var dataset = [ 1, 2, 3, 4, 5 ];

d3.select("body").selectAll("div")
    .data(dataset)
    .enter()
    .append("div")
    .attr("class", "bar");
```

- above sample outputs the values from our dataset with no space between them
  - *effectively as a bar chart of equal height*
- modify the height of each representative bar
  - *by setting height of each bar as a function of its corresponding data value*
  - *eg: append the following to our example chain*

```
.style("height", function(d) {
    return d + "px";
});
```

- make each bar in our chart more clearly defined by modifying `style`

```
.style("height", function(d) {
    var barHeight = d * 3;
    return barHeight + "px";
});
```

Data visualisation - D3

Drawing - intro - part 3

1. drawing SVGs

- properties of SVG elements are specified as **attributes**
- represented as property/value pairs within each element tag

```
<element property="value">...</element>
```

- SVG elements exist in the DOM
  - *we can still use D3 methods `append()` and `attr()`*
  - *create new HTML elements and set their attributes*

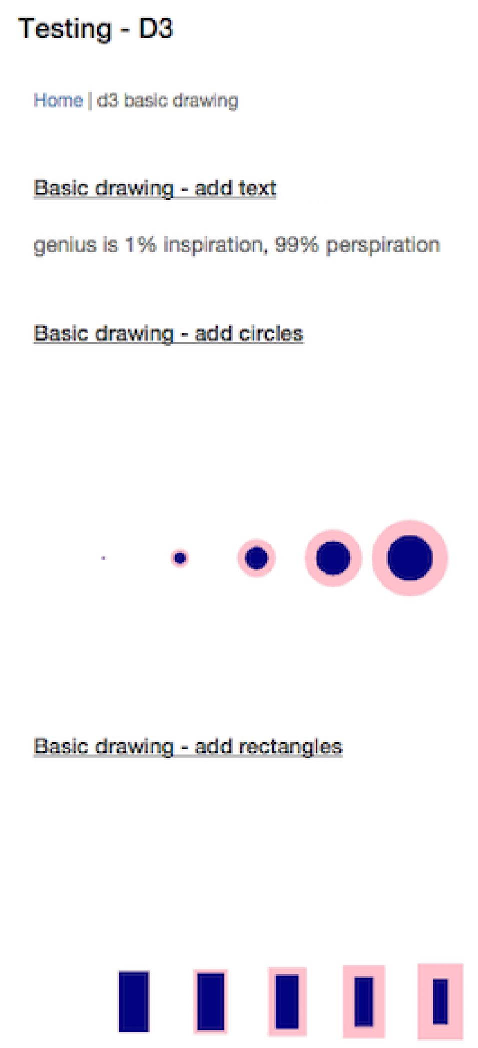
2. create SVG

- need to create an element for our SVG
- allows us to draw and output all of our required shapes

```
d3.select("body").append("svg");
```

- variable effectively works as a reference
  - *points to the newly created SVG object*
  - *allows us to use this reference to access this element in the DOM*
- DEMO - Drawing with SVG

Image - D3 Basic Drawing



D3 - basic drawing

Data visualisation - D3

Drawing - SVG barchart - part 1

- create a new barchart using SVG, need to set the required size for our SVG output

```
//width & height
var w = 750;
var h = 200;
```

- then use D3 to create an empty SVG element, and add it to the DOM

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

- instead of creating DIVs as before, we generate *rects* and add them to the *svg* element.

```
svg.selectAll("rect")
    .data(dataset)
    .enter()
    .append("rect")
    .attr("x", 0)
    .attr("y", 0)
    .attr("width", 10)
    .attr("height", 50);
```

Data visualisation - D3

Drawing - SVG barchart - part 2

- this code selects all of the `rect` elements within `svg`
- initially none, D3 still needs to select them before creating them
- `data()` then checks the number of values in the specified dataset
  - *hands those values to the `enter` method for processing*
- `enter` method then creates a placeholder
  - *for each data value without a corresponding `rect`*
  - *also appends a rectangle to the DOM for each data value*
- then use `attr` method to set `x, y, width, height` values for each rectangle
- still only outputs a single bar due to an overlap issue
- need to amend our code to handle the width of each bar
  - *implement flexible, dynamic coordinates to fit available SVG width and height*
  - *visualisation scales appropriately with the supplied data*

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

Data visualisation - D3

Drawing - SVG barchart - part 3

- now linked the `x` value directly to the width of the SVG `w`
  - *and the number of values in the dataset, `dataset.length`*
  - *the bars will be evenly spaced regardless of the number of values*
- if we have a large number of data values
  - *bars still look like one horizontal bar*
  - *unless there is sufficient width for parent SVG and space between each bar*
- try to solve this as well by setting the bar width to be proportional
  - *narrower for more data, wider for less data*

```
var w = 750;
var h = 200;
var barPadding = 1;
```

- now set each bar's width
  - *as a fraction of the SVG width and number of data points, minus our padding value*

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

- our bar widths and x positions scale correctly regardless of data values

Data visualisation - D3

Drawing - SVG barchart - part 4

- encode our data as the *height* of each bar

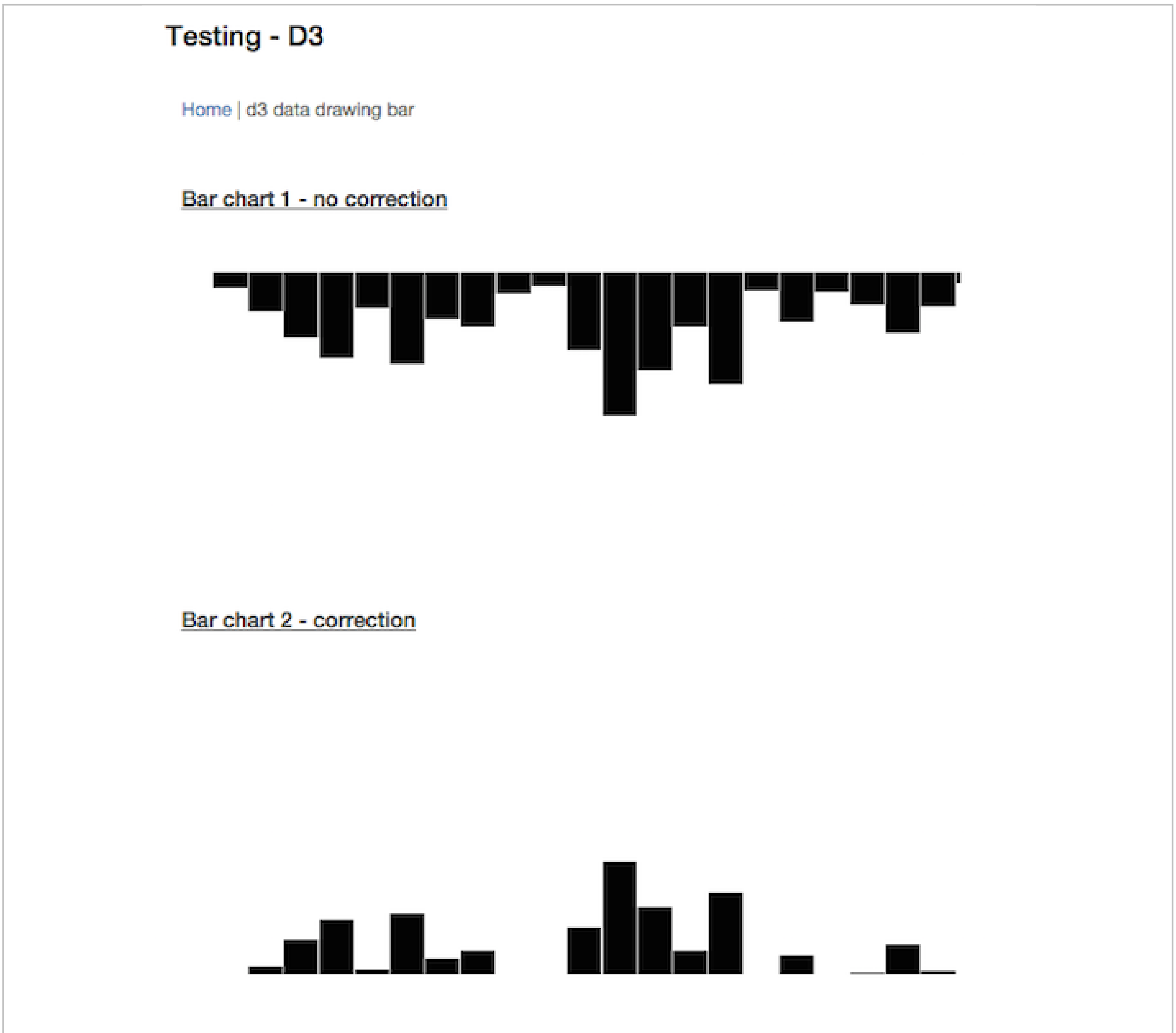
```
.attr("height", function(d) {
    return d * 4;
});
```

- our bar chart will size correctly, albeit from the top down
  - *due to the nature of SVG*
  - *SVG adheres to a top left pattern for rendering shapes*
- to correct this issue
  - *need to calculate the top position of our bars relative to the SVG*
- top of each bar expressed as a relationship
  - *between the height of the SVG and the corresponding data value*

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

- bar chart will now display correctly from the bottom upwards
- DEMO - Drawing with SVG - barcharts

Image - D3 Barcharts



D3 - drawing barcharts

Data visualisation - D3

Drawing - SVG barchart - part 5

1. add some colour

- adding a colour per bar simply a matter of setting an attribute for the fill colour

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

- set many colours using the data itself to determine the colour

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

2. add text labels

- also set dynamic text labels per bar, which reflect the current dataset

```
svg.selectAll("text")
.data(dataset)
.enter()
.append("text")
```

- extend this further by positioning our text labels

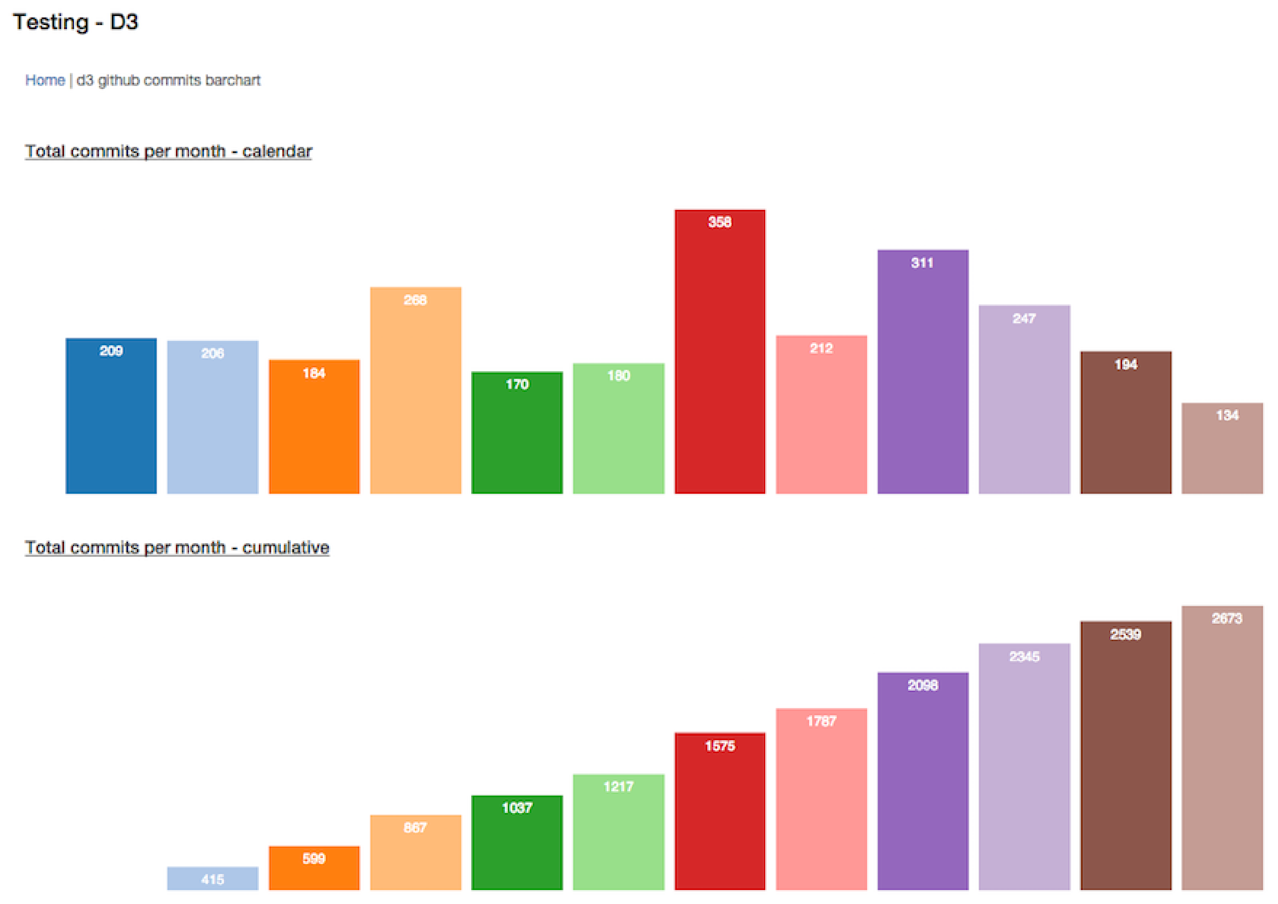
```
.attr("x", function(d, i) {
    return i * (w / dataset.length);
})
.attr("y", function(d, i) {
    return h - (d * 4);
});
```

- then position them relative to the applicable bars, add some styling, colours...

```
.attr("font-family", "sans-serif")
.attr("font-size", "11px")
.attr("fill", "white");
```

- DEMO - Drawing with SVG - barcharts, colour, and text labels

Image - D3 Barcharts



D3 - drawing barcharts with colour and text

Data visualisation - D3

Drawing - add interaction - listeners

- event listeners apply to any DOM element for interaction
    - *from a button to a <p> with the body of a HTML page*

```
<p>this is a HTML paragraph...</p>
```

- add a listener to this DOM element

```
d3.select("p")
    .on("click", function() {
    //do something with the element...
    });
```

- above sample code selects the <p> element
    - *then adds an event listener to that element*
- *event listener* is an anonymous function
    - *listens for `.on` event for a specific element or group of elements*
- in our example,
    - *`on()` function takes two arguments*

Data visualisation - D3

Drawing - add interaction - update visuals

- achieved by combining
  - *event listener*
  - *modification of the visuals relative to changes in data*

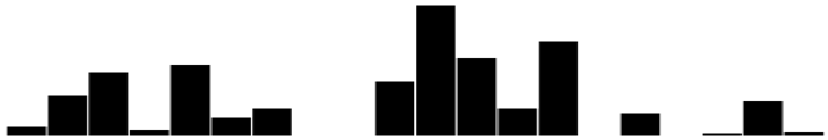
```
d3.select("p")
    .on("click", function() {

    dataset = [....];

    //update all of the rects
    svg.selectAll("rect")
    .data(dataset)
    .attr("y", function(d) {
    return h - yScale(d);
    });
    .attr("height", function(d) {
    return yScale(d);
    });
});
```

- above code triggers a change to visuals for each call to the event listener
- eg: change the colours
  - *add call to `fill()` to update bar colours*

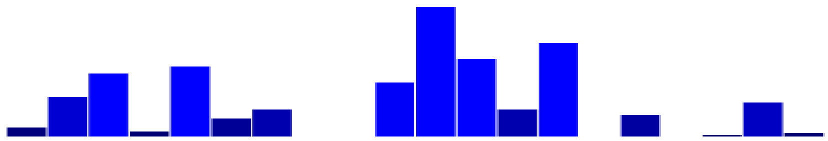
```
.attr("fill", function( d) {
    return "rgb( 0, 0, " + (d * 10) + ")";
});
```

- DEMO - update bar colours

Image - D3 Barcharts



Bar chart 3 - colours



D3 - drawing colour updates for barcharts

Data visualisation - D3

Drawing - add interaction - transitions

- adding a fun transition in D3 is as simple as adding the following,

```
.transition()
```

- add this to above code chain to get a fun and useful transition in the data
- animation reflects the change from the old to the new data
- add a call to the `duration()` function
  - *allows us to specify a time delay for the transition*
  - *quick, slow...we can specify each based upon time*
- chain the `duration()` function after `transition()`

```
.transition().duration(1000)
```

- if we want to specify a constant easing to the transition
  - *use `ease()` with a `linear` parameter*

```
.ease(linear)
```

- other built-in options, including
  - *circle - gradual ease in and acceleration until elements snap into place*
  - *elastic - best described as springy*
  - *bounce - like a ball bouncing, and then coming to rest...*

Data visualisation - D3

Drawing - add interaction - transitions

- add a delay using the `delay()` function

```
.transition()
.delay(1000)
.duration(2000)
```

- also set the `delay()` function dynamically relative to the data,

```
.transition()
.delay( function( d, i) {
return i * 100;
})
.duration( 500)
```

- when passed an anonymous function
  - *datum bound to the current element is passed into `d`*
  - *index position of that element is passed into `i`*
- in the above code example, as D3 loops through each element
  - *delay for each element is set to `i * 100`*
  - *meaning each subsequent element will be delayed 100ms more than preceding element*
- DEMO - transitions - interactive sort

Data visualisation - D3

Drawing - add interaction - adding values and elements

- select all of the bars in our chart
  - *we can rebind the new data to those bars*
  - *and grab the new update as well*

```
var bars = svg.selectAll("rect")
    .data(dataset);
```

- if more new elements, bars in our example, than original length
  - *use `enter()` to create references to those new elements that do not yet exist*
- with these reserved elements
  - *we can use `append()` to add those new elements to the DOM*
  - *now updates our bar chart as well*
- now made the new `rect` elements
  - *need to update all visual attributes for our `rects`*
  - *set x, and y position relative to new dataset length*
  - *set width and height based upon new xScale and yScale*
  - *calculated from new dataset length*

Data visualisation - D3

Drawing - add interaction - removing values and elements

- more DOM elements than provided data values
  - *D3's **exit** selection contains references to those elements without specified data*
  - ***exit** selection is simply accessed using the `exit()` function*
- grab the *exit* selection
- then transition exiting elements off the screen
  - *for example to the right*
- then finally remove it

```
bars.exit()
.transition()
.duration(500)
.attr("x", w)
.remove();
```

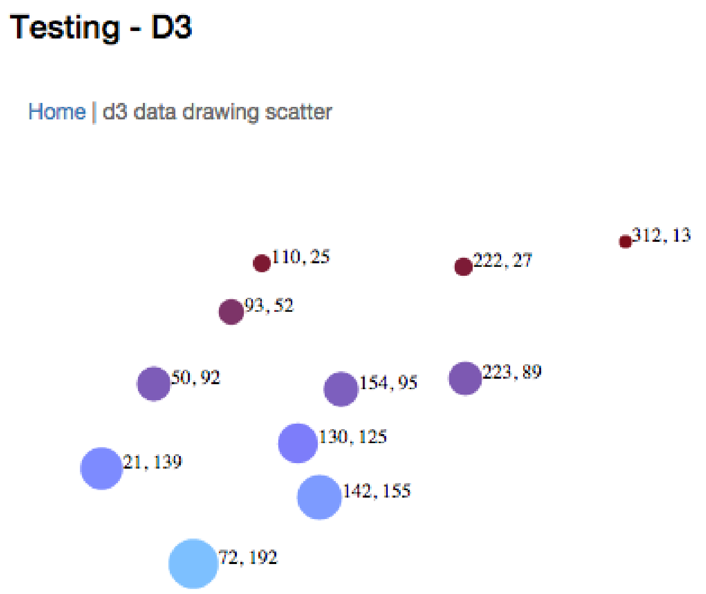
- `remove()` is a special transition method that awaits until transition is complete
- then deletes element from DOM forever
  - *to get it back, we'd need to rebuild it again*

Data visualisation - D3

Drawing - SVG scatterplot - intro

- scatterplot allows us to visualise two sets of values on two different axes
  - *one set of data against another*
- plot one set of data on *x* axis, and the other on the *y* axis
- often create dimensions from our data
  - *helps us define patterns within our dataset*
  - *eg: date against age, or age against fitness...*
- dimensions will also be represented relative to *x* and *y* axes
- create our scatterplot using SVG
  - *add our SVG to a selected element*

Image - D3 Scatterplot



D3 - drawing a basic scatterplot

Data visualisation - D3

Drawing - SVG scatterplot - data

- data for the scatterplot is normally stored as a multi-dimensional representation
  - *comparison x and y points*
- eg: we could store this data in a multi-dimensional array

```
var dataset = [
        [10, 22], [33, 8], [76, 39], [4, 15]
      ];
```

- in such a multi-dimensional array
  - *inner array stores the comparison data points for our scatterplot*
  - *each inner array stores x and y points for scatterplot diagram*
- we can also stroe such data in many different structures
  - *eg: JSON...*

Data visualisation - D3

Drawing - SVG scatterplot - create SVG

- need to create an element for our SVG
  - *allows us to draw and output all of our required shapes*

```
d3.select("body").append("svg");
```

- appends to the body an SVG element
  - *useful to encapsulate this new DOM element within a variable*

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

- variable effectively works as a reference
  - *points to the newly created SVG object*
  - *allows us to use this reference to access element in the DOM*

Data visualisation - D3

Drawing - SVG scatterplot - build scatterplot

- as with our barchart, we can set the width and height for our scatterplot,

```
//width & height
var w = 750;
var h = 200;
```

- we will need to create circles for use with scatterplot instead of rectangles

```
svg.selectAll('circle')
   .data(dataset)
   .enter()
   .append('circle');
```

- corresponding to drawing circles
  - *set `cx`, the x position value of the centre of the circle*
  - *set `cy`, the y position value of the centre of the circle*
  - *set `r`, the radius of the circle*

Data visualisation - D3

Drawing - SVG scatterplot - adding circles

- draw circles for scatterplot

```
.attr('cx', function(d) {
    return d[0]; //get first index value for inner array
})
.attr('cy', function(d) {
    return d[1]; //get second index value for inner array
})
.attr('r', 5);
```

- outputs simple circle for each inner array within our supplied multi-dimensional dataset
- start to work with creating circle sizes relative to data quantities
- set a dynamic size for each circle
  - *representative of the data itself*
  - *modify the circle's area to correspond to its y value*
- as we create SVG circles, we cannot directly set the area
  - *so we need to calculate the radius* `r`
  - *then modify that for each circle*

Data visualisation - D3

Drawing - SVG scatterplot - calculate dynamic area

- assuming that `d[1]` is the original area value of our circles
  - *get the square root and set the radius for each circle*
- instead of setting each circle's radius as a static value
  - *now use the following*

```
.attr('r', function(d) {
    return Math.sqrt(d[1]);
});
```

- use the JavaScript `Math.sqrt()` function to help us with this calculation

Data visualisation - D3

Drawing - SVG scatterplot - add colour

- as with a barchart
- also set a dynamic colour relative to a circle's data

```
.attr('fill', function (d) {
    return 'rgb(125,' + (d[1]) + ', ' + (d[1] * 2) + ')';
});
```

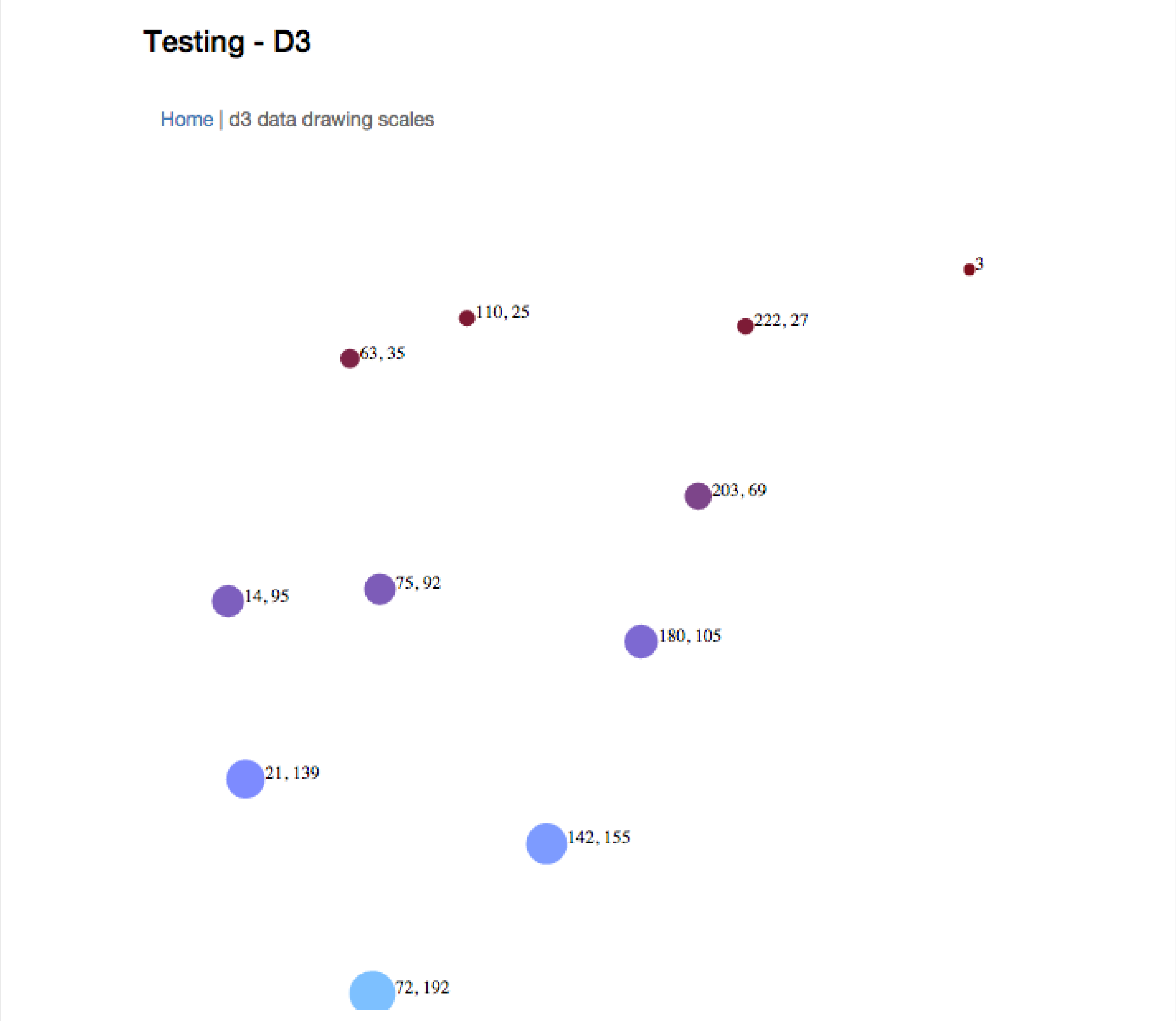
Data visualisation - D3

Drawing - SVG scatterplot - add labels

```
//add labels for each circle
svg.selectAll('text')
   .data(dataset)
   .enter()
   .append('text')
   .text(function(d) {
    return d[0] + ', ' + d[1];//set each data point on the text label
   })
   .attr('x', function(d) {
    return d[0];
   })
   .attr('y', function(d) {
    return d[1];
   })
   .attr('font-family', 'serif')
   .attr('font-size', '12px')
   .attr('fill', 'navy');
```

- start by adding text labels for our data
  - *adding new text elements where they do not already exist*
- then set the text label itself for each circle
  - *using the data values stored in each inner array*
- make the label easier to read
  - *set x and y coordinates relative to data points for each circle*
- set some styles for the labels

Image - D3 Scatterplot



D3 - drawing a basic scatterplot 2

Data visualisation - D3

Drawing - SVG - scales

- in D3, *scales* are defined as follows,

> *"Scales are functions that map from an input domain to an output range"*
>
> *Bostock, M.*

- you can specify your own scale for the required dataset
  - *eg: to avoid massive data values that do not translate correctly to a visualisation*
  - *scale these values to look better within you graphic*
- to achieve this result, you simply use the following pattern.
  - *define the parameters for the scale function*
  - *call the scale function*
  - *pass a data value to the function*
  - *the scale function returns a scaled output value for rendering*
- also define and use as many scale functions as necessary for your visualisation
- important to realise that a scale has no direct relation to the visual output
  - *it is a mathematical relationship*
- need to consider scales and axes
  - *two separate, different concepts relative to visualisations*

Data visualisation - D3

Drawing - SVG - domains and ranges

- *input domain* for a scale is its possible range of input data values
  - *in effect, initial data values stored in your original dataset*
- *output range* is the possible range of output values
  - *normally use as the pixel representation of the data values*
  - *a personal consideration of the designer*
- normally set a minimum and maximum *output range* for our scaled data
- scale function then calculates the scaled output
  - *based upon original data and defined range for scaled output*
- many different types of scale available for use in D3
- three primary types
  - *quantitative*
  - *ordinal*
  - *time*
- *quantitative* scale types also include other built-in scale types
- many methods available for the scale types

Data visualisation - D3

Drawing - SVG - building a scale

- start building our scale in D3
  - *use `d3.scale` with our preferred scale type*

```
var scale = d3.scale.linear();
```

- to use the scale effectively, we now need to set our input domain

```
scale.domain([10, 350]);
```

- then we set the output range for the scale

```
scale.range([1, 100]);
```

- we can also chain these methods together

```
var scale = d3.scale.linear()
          .domain([10, 350])
              .range([1, 100]);
```

Data visualisation - D3

Drawing - SVG - adding dynamic scales

- we could pre-define values for our scale relative to a given dataset
- makes more sense to abstract these values relative to the defined dataset
- we can now use the D3 array functions to help us set these scale values
  - *eg; find highest number in array dataset*

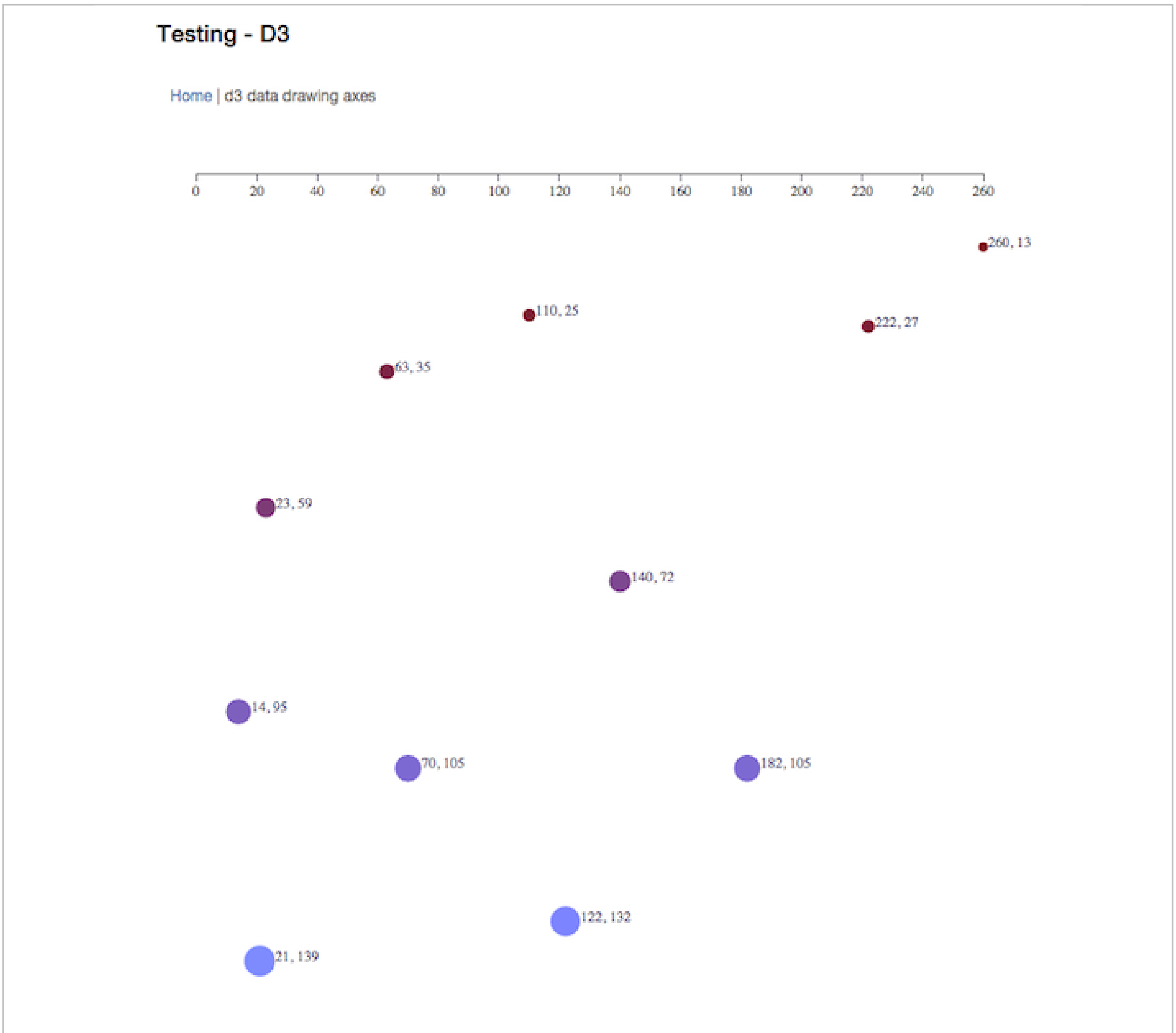
```
d3.max(dataset, function(d) {
    return d[0];
});
```

- returns highest value from the supplied array
- getting minimum value in array works in the same manner
  - *with `d3.min()` being called instead*
- now create a scale function for `x` and `y` axes

```
var scaleX = d3.scale.linear()
        .domain([0, d3.max(dataset, function(d) { return d[0]; })])
        .range([0, w]);//set output range from 0 to width of svg
```

- Y axis scale modifies above code relative to provided data, `d[1]`
  - *range uses height instead of width*
- for a scatterplot we can use these values to set `cx` and `cy` values

Image - D3 Scatterplot



D3 - add axis

Data visualisation - D3

Drawing - SVG - adding dynamic scales

- a few data visualisation examples
- Tests 1
- Tests 2

general examples

Sample dashboards and visualisations

- gaming dashboard
- schools and education
- students and grades
- D3 examples

Example datasets

- Chicago data portal

Article example

- dashboard designs
- replace jQuery with D3

projects examples

A few examples from recent projects,

- GitHub API tests
- check JSON return
- early test examples
- metrics test examples

Text Encoding Initiative - general usage

Exercise - part 1

```
<TEI>
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>
          <!--Title-->
        </title>
      </titleStmt>
      <publicationStmt>
        <p>
          <!--Publication Information-->
        </p>
      </publicationStmt>
      <sourceDesc>
        <p>
          <!--Information about the source-->
        </p>
      </sourceDesc>
    </fileDesc>
  </teiHeader>
  <text>
    <body>
      <!--Some structural division, paragraph, line group, speech, ...-->
    </body>
  </text>
</TEI>
```

- using the above template
  - *add some of the missing information - as commented in the XML*
  - *write a simple XSL stylesheet to render the content and metadata to HTML*
  - *test in a browser*
- ~ 15 minutes

Text Encoding Initiative - general usage

- within the <teiHeader> we could add
  - *the title of the text*
  - *<respStmt>*
  - *<publicationStmt>*
  - *<sourceDesc>*
- within <text> we could add
  - *<front> with a dedication and table of contents...*
  - *<body> with transcription etc of the text*
  - *<back> with a div of type 'colophon'...*

Text Encoding Initiative - general usage

<teiHeader> example

```xml
<teiHeader>
  <fileDesc>
    <titleStmt>
      <title>The Epic of Gilgamesh</title>
      <respStmt>
        <resp>editor</resp>
        <name xml:id="AL">Alasdair Livingstone</name>
      </respStmt>
    </titleStmt>
    <publicationStmt>
      <p>Not for distribution.</p>
    </publicationStmt>
    <sourceDesc>
      <p>Transcribed from the diaries of Professor Wilfred Lambert</p>
    </sourceDesc>
  </fileDesc>
</teiHeader>
```

Text Encoding Initiative - general usage

<text> example

```
<text>
  <front>
    <div type="dedication">
      <p>personal dedication...</p>
    </div>
    <div type="contents">
      <head>Table of Contents</head>
      <list>
        <item>1. No.1...</item>
        <item>2. No.2...</item>
        <item>3. No.3...</item>
      </list>
    </div>
 </front>
 <body>
    <p>some body text goes here....</p>
  </body>
 <back>
    <div type="colophon">
      <p>Physical book conditions...</p>
    </div>
 </back>
</text>
```

Text Encoding Initiative - textual phenomena

structural features

- Prose elements, e.g.

```
<p> <div> <head> <list> <item> <q> <pb> <seg> <figure> <tables>
```

- Verse elements, e.g.

```
<lg> <l>
```

- Drama elements, e.g.

```
<div> <sp> <p> <lg> <l> <seg>
```

Text Encoding Initiative - textual phenomena

structural features - examples

- list

```
<list>
  <item></item>
</list>
```

- line groupings

```
<lg type="free">
  <l></l>
</lg>
```

- segments

```
<seg type="preamble">
  <seg></seg>
  <seg></seg>
</seg>
```

- quotations

```
<q></q>
```

- an example with 'quoted' q

Text Encoding Initiative - textual phenomena

title pages

- <titlePage> used for transcription and encoding of physical title page
- within <front> or <back> using <titlePage>
- <titlePage> commonly includes <docTitle>
- <docTitle> may include subsections, <titlePart>
- <docAuthor> for name of the author
- <byline> for primary statement of responsibility for a work
- other elements can include
  - *<docEdition>*
  - *<docDate>*
  - *<docImprint> may contain*
  - *<pubPlace>*
  - *<docDate>*
  - *<publisher>*
  - ...
  - *<epigraph>*
  - *<imprimatur>*
  - *<graphic/>*

Text Encoding Initiative - textual phenomena

title pages - examples

- <docTitle> can consist of several subsections
- <titlePart>, which may itself use a @type attribute documenting their role.

```
<docTitle>
  <titlePart type="main">main title...</titlePart>
  <titlePart type="sub">sub-title</titlePart>
</docTitle>
```

- an inline graphic, illustration, or figure <graphic/>

```
<figure>
 <graphic url="fig1.png"/>
 <head>.</head>
 <figDesc>.</figDesc>
</figure>
```

Text Encoding Initiative - general usage

Exercise - part 2 - Front & Title Page with Image

Use the image on the next slide to encode a title page in the front part of a text.

Encode the following:

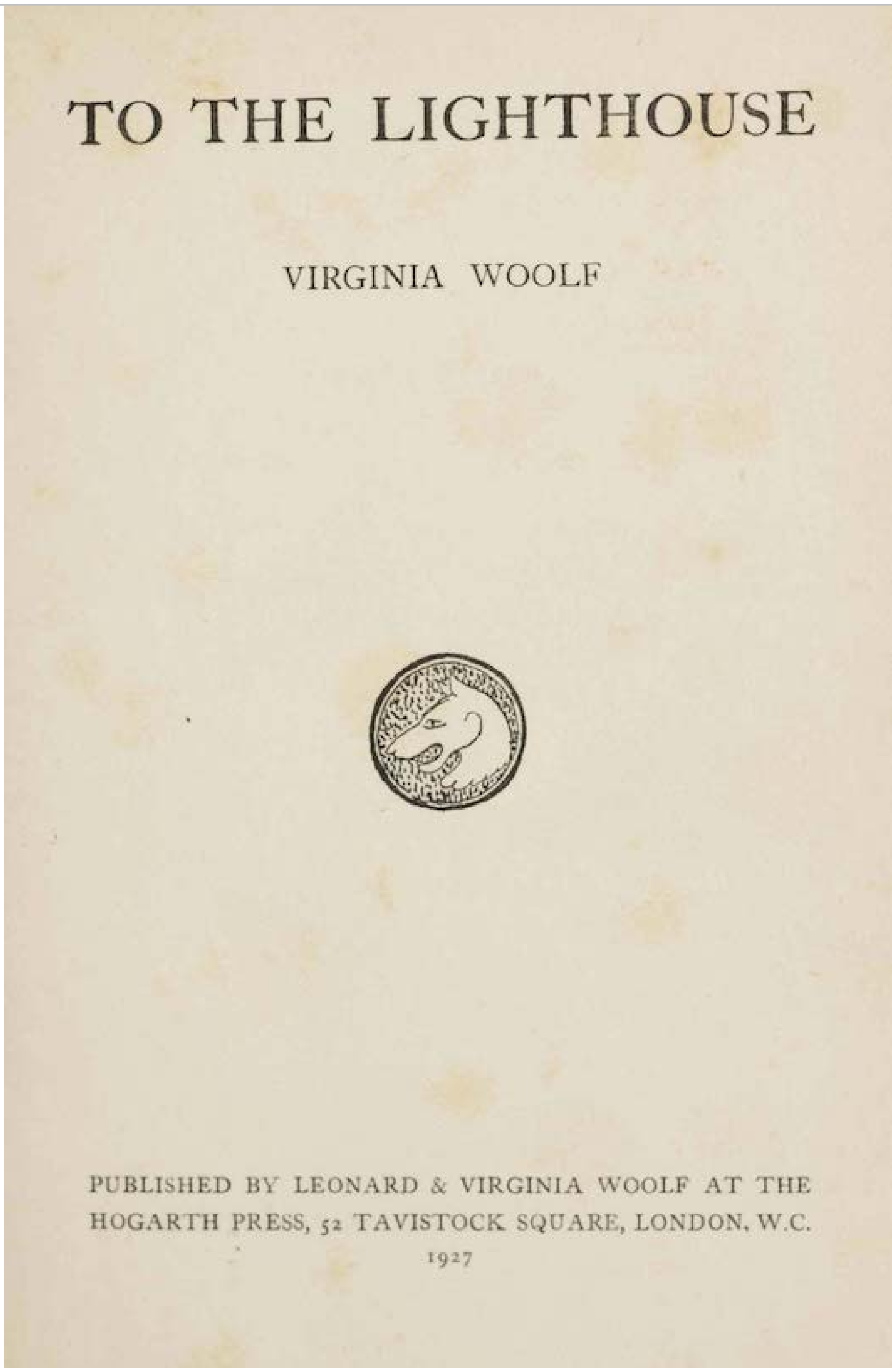
1. title and author information
2. any stylistic considerations in the font, size, spacing...
3. the graphic
4. any other appropriate information such as edition, publication information...

Please feel free to consult the TEI guidelines where necessary.

- ~ 15 minutes

Image - To The Lighthouse

front page



TO THE LIGHTHOUSE

VIRGINIA WOOLF

PUBLISHED BY LEONARD & VIRGINIA WOOLF AT THE HOGARTH PRESS, 52 TAVISTOCK SQUARE, LONDON, W.C.
1927

To the Lighthouse - GB 1st Edition

front & title page example

A simple example of encoding a front and title page

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title/>
      </titleStmt>
      <publicationStmt>
        <publisher/>
      </publicationStmt>
      <sourceDesc>
        <p/>
      </sourceDesc>
    </fileDesc>
  </teiHeader>
  <text>
    <front>
      <titlePage>
        <docTitle>
          <titlePart type="main">TO THE LIGHTHOUSE</titlePart>
        </docTitle>
        <byline>
          <docAuthor>Virginia Woolf</docAuthor>
        </byline>
        <figure>
          <graphic url="hogarth-logo.jpg"/>
          <figDesc>Black and white rendition of Hogarth Logo</figDesc>
        </figure>
        <docImprint>
          <publisher>PUBLISHED BY LEONARD &AMP; VIRGINIA WOOLF AT THE HOGARTH PRE
          <pubPlace>52 TAVISTOCK SQUARE, LONDON. W.C.</pubPlace>,
          <docDate>1927</docDate>
        </docImprint>
      </titlePage>
    </front>
    <body>
    </body>
```

```
  </text>
</TEI>
```

n.b. in the <docImprint> element

- for the <publisher> element - we could be more specific and use <name> elements
- e.g. for Leonard and Virginia Woolf
- for the <pubPlace> element - we could enclose this within an <address> element
- this might help to add further structure and metadata
- for the <docDate> element - we could add a @when attribute with value set to the year 1927
- again, it helps to add further metadata

Text Encoding Initiative - textual phenomena

logical and semantic features

- encodes underlying semantics with specific elements other than <hi>
- highlighting may be used to show

```
<emph> <foreign> <distinct>
```

- uses of quotation

```
<said> <quote> <cit> <mentioned> <soCalled>
```

- other examples include

```
<term> <gloss>
```

- generic elements may also carry semantic and logical information

```
<title> <name> <num> <measure> <date> <address> <abbr> <expan>
```

Text Encoding Initiative - textual phenomena

logical and semantic features - referring strings

- in TEI
  - *places, people, objects can use the `<name>` elements with `@type` attribute*
  - e.g.

```
<name type="person">Jack</name>
```

- proper nouns can also be referred to using the `<rs>` element plus `@type` attribute
  - e.g.

```
<rs type="person">he</rs>
```

Text Encoding Initiative - general usage

Exercise - part 3 - common structure and elements

Use the excerpt on the next slide to encode a structured grouping of text, e.g. a division, segment, page...

Encode the following:

1. this text as a paragraph of prose
2. indicate that the language of this paragraph is English
3. indicate that the paragraph is number 7
4. encode any abbreviations with their correct expansions
5. encode names with the appropriate element and attribute

Please feel free to consult the TEI guidelines where necessary.

- ~ 15 minutes

Text Encoding Initiative - general usage

Exercise - part 3 - text excerpt

- encode the following text excerpt

```
  "Good-evening, Mrs. McNab," she would say.
She had a pleasant way with her. The girls
all liked her. But dear, many things had changed
since then (she shut the drawer); many families
had lost their dearest. So she was dead; and
Mr. Andrew killed; and Dr. Prue dead too,
they said, with her first baby; but every one had
lost some one these years. Prices had gone up
shamefully, and didn't come down again neither.
She could well remember her in her grey cloak.
```

Text Encoding Initiative - general usage

text excerpt - element encoding of local language

A simple example of a text excerpt with element defined local language encoding.

```xml
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title/>
      </titleStmt>
      <publicationStmt>
        <publisher/>
      </publicationStmt>
      <sourceDesc>
        <p/>
      </sourceDesc>
    </fileDesc>
  </teiHeader>
  <text>
    <body>
      <p xml:lang="en" n="7">"Good-evening,
        <name type="person">
          <choice>
            <abbr type="title">Mrs.</abbr>
            <expan>Mistress</expan>
          </choice>
        McNab</name>
      ," she would say.</p>
    </body>
  </text>
</TEI>
```

n.b.

- `en-GB` - the value must conform to BCP 47 (tags for identifying languages)

- the `<rs>` element with a `@type` attribute can also be used instead of `<name>`
- `<name>` is solely for proper nouns
- the rest of the paragraph in the example follows the same pattern

text excerpt - header encoding of global language

A simple example of a text excerpt with header defined global language encoding.

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title/>
      </titleStmt>
      <publicationStmt>
        <publisher/>
      </publicationStmt>
      <sourceDesc>
        <p/>
      </sourceDesc>
    </fileDesc>
    <profileDesc>
      <langUsage>
        <language ident="en-GB">British English</language>
      </langUsage>
    </profileDesc>
  </teiHeader>
  <text>
    <body>
      <p n="7"[5]>"Good-evening,
        <name type="person">
          <choice>
            <abbr type="title">Mrs.</abbr>
            <expan>Mistress</expan>
          </choice>
        McNab</name>
      ," she would say.</p>
    </body>
  </text>
</TEI>
```

n.b.

- header encoding of global language this helps specify the languages used throughout the text being encoded
- we could specify the language used, as in the above example, but now it would only make sense if the language was different from the specified default in the TEIHeader.

Text Encoding Initiative - textual phenomena

logical and semantic features - dates and time

- `<date>` and `<time>`
- system or calendar used may be documented using `@calendar` attribute
- value of date or time supplied using `@when` attribute
- normalised representation of `<date>` should conform to valid W3C datatype

```
<date when="2003-12-22" calendar="Gregorian">22 Nov 2003</date>
```

- `<date>` can also be used to mark a period of time using the attributes
  - *@from*
  - *@to*
  - *@notBefore*
  - *@notAfter*

Text Encoding Initiative - textual phenomena

logical and semantic features - numbers and measures

- <num>

```
<num type="percentage" value="23">23%</num>
```

- <measure>

```
<measure type="volume" quantity="1.5" unit="litre" commodity="wine">1.5L bottle o
```

Exercise - part 4

If **20,000 Leagues Under the Sea** was an actual recording of the distance travelled by Captain Nemo in the Nautilus, how would you specify this using TEI?

n.b. league = ~ 3 nautical miles or 3.5 standard English miles

- create a TEI encoded XML document
  - *add some details for the novel **20,000 Leagues Under the Sea** by **Jules Verne***
  - *correctly encode **20,000 leagues***
  - *write a simple XSL stylesheet to render the content and metadata to HTML*
  - *test in a browser*
- ~ 15 minutes

Text Encoding Initiative - textual phenomena

logical and semantic features - addresses

- physical and digital addresses can be encoded
  - *<address>*, *<email>*
- `<email>` can also use a `@type` attribute
- `<addrLine>` within `<address>`
  - *<name>*, *<street>*, *<postCode>*, *<postBox>*

Text Encoding Initiative - textual phenomena

addresses - examples

```
<address>
  <addrLine></addrLine>
</address>
```

- you can also be specific, and use semantically rich elements

```
<name>
<street>
<postCode>
<postBox>
```

Text Encoding Initiative - textual phenomena

Exercise - part 5

- how would you encode **Alice's** address in **Wonderland** using TEI?
- create example TEI XML and encode this **fictional** address
- write a simple XSL stylesheet to render the content and metadata to HTML
- test in a browser
- ~ 10 minutes

Text Encoding Initiative - textual phenomena

logical and semantic features - abbreviations and expansions

- explicitly encode
  - *<abbr>*
  - *<expan>*
- `@type` attribute may be used with `<abbr>`
  - e.g.

```
<abbr type="title">Dr</abbr>
```

- `<expan>` used with `<abbr>` within `<choice>`

Text Encoding Initiative - textual phenomena

abbreviations and expansions - example

```
<choice>
  <abbr type="title">Dr</abbr>
  <expan>Doctor</expan>
</choice>
```

Text Encoding Initiative - textual phenomena

Exercise - part 6

- how might you use the `choice` element to encode **PhD**?
- use TEI encoded XML
- ~ 10 minutes

Text Encoding Initiative - textual phenomena

analytical features - notes and annotations

- <note> can be used to record a textual annotation
- @type attribute used to differentiate notes
- @resp attribute used to assign responsibility
- position of the note can be referenced using the @place attribute

e.g.

```
<note n="1" place="foot" type="editorial" resp="NJH">a new note...</note>
```

Text Encoding Initiative - textual phenomena

analytical features - index entries

- pre-existing in `<front>` or `<back>`
  - *using `<list>` inside `<div>`*
- new index using `<term>` inside `<index>`
  - *add at the location of the index item*
  - e.g.

```
<index>
  <term>new index term...</term>
</index>
```

Text Encoding Initiative - textual phenomena

analytical features - errors

- indicated using `<sic>`
- corrected using `<corr>`
- combine `<sic>` and `<corr>` within `<choice>`
- use `@cert`and `@resp` attributes to encode degree of certainty and editor responsible
- e.g.

```
<corr cert="high" resp="#NJH">correction...</corr>
```

- **n.b.** *hash* in the `@resp` attribute value
  - *a pointer to a name element in the <teiHeader>*
  - e.g.

```
<name xml:id="NJH">...</name>
```

Text Encoding Initiative - textual phenomena

Exercise - part 7

- how would you add a pointer to a name element in the `<teiHeader>`?
- use TEI encoded XML
- ~ 10 minutes

<respStmt> - example

- place this name within the <respStmt> element as follows,

```
<respStmt>
  <resp>editor</resp>
  <name xml:id="NJH">Nicholas J Hayward</name>
</respStmt>
```

- then reference this editor as necessary in the encoded document

Text Encoding Initiative - textual phenomena

analytical features - regularisation

- <reg> element for regularisation
- <orig> for original, non-normalised form
- use <reg> in isolation or combined with <orig> within <choice>
- e.g.

```
<choice>
  <orig>thou</orig>
  <reg resp="#NJH">you</reg>
</choice>
```

Text Encoding Initiative - textual phenomena

analytical features - additions, deletions & omissions

- <gap> element used for omission, both material and editorial
- @reason attribute used to indicate reason for omission
- @extent and @unit attributes can be used to record extent of omission
- editorial omissions should be recorded using <editorialDecl>
  - *add inside <editorialDesc> in <teiHeader>*
- <gap> may be empty or include a <desc> of the material omitted
- <add> and <del> may also be used for words and phrases
- @rend attribute may also be used with <add> and <del>
- <addSpan/> and <delSpan/> for longer passages
- <subst> to contain <add> and <del> with causal relationship
- <unclear> with @reason attribute for difficult to read deletions in the text

Text Encoding Initiative - non-textual phenomena

graphics &c.

- graphics such as illustrations, diagrams, drawings, artwork...
- anchor in a text using `<graphic/>` and optional `@url` attribute
- e.g.

```
<graphic url="http://www.somewhere.com/image.jpg"/>
```

- use `<figure>` element as parent to create a full listing for a graphic

Text Encoding Initiative - Attributes

global Attributes

Global Attributes are currently as follows:

- @cert
  - *provided by `responsibility` subclass*
- @n
- @rend
  - *provided by `rendition` subclass*
- @rendition
  - *provided by `rendition` subclass*
- @resp
  - *provided by `responsibility` subclass*
- @source
  - *provided by `source` subclass*
- @style
  - *provided by `rendition` subclass*
- @xml:base
- @xml:id
- @xml:lang
- @xml:space

Text Encoding Initiative - Further Examples

bibliography listing

- Sample from Verne Digital Corpus

Text Encoding Initiative - Further Examples

regularise

- regularise with soundex and metaphone
- regularise with soundex and metaphone - basic compare
- regularise with soundex and metaphone - array compare

Text Encoding Initiative - Further Examples

working with images

- working with image and transcription
  - *representation of primary sources*
- TEI
  - *Facsimile - guidelines & examples*
  - *Surface - guidelines & examples*
  - *Zone - guidelines & examples*