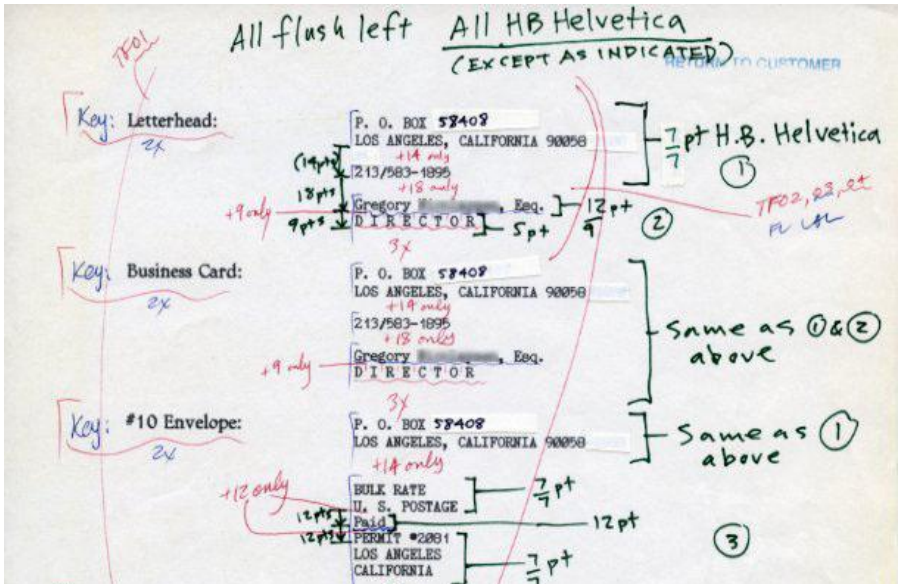
Software Design and Construction
159.251
Markup Languages

Amjed Tahir

a.tahir@massey.ac.nz

Based on slides from Shawn Rasheed

A brief history of markup



TeX, PostScript,

SGML, HTML, XML, XHTML

Image: https://www.marksimonson.com/notebook/view/the-lost-art-of-type-specing

Primer on computer languages

Computer languages express data or computation.

- **General purpose languages** (Java, Python, C++)
- **Domain Specific Languages (DSL)** (SQL, HTML, TeX/LaTeX, PostScript)
- **Data languages** (JSON, YAML, XML)

Rule of least power: Use the least powerful language for a particular purpose.

Expressive power of a language

  - **Theoretical (Turing-completeness)**
  - **Practical (Concise, verbose)**

Expressing data

- Data can be
  - self-describing (metadata as part of the data)
  - Non-self-describing. E.g. XDR (extended detection and response)

- Types:

  - Structured data: i.e., relational / tabular data (SQL)
  - Unstructured/semi-structured data: i.e., markup languages (method to convey metadata within a document) (XML, YAML, JSON)

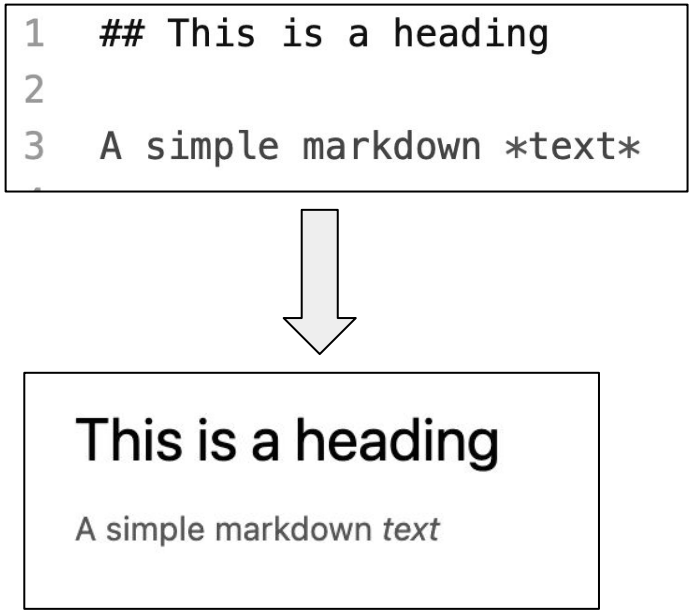
Types of markup

- **Presentational/punctuational markup:** using content such as punctuation, indentation, whitespace to denote structure in a document (Word processors).

  It produces WYSIWYG files → "what you see is what you get"

Types of markup

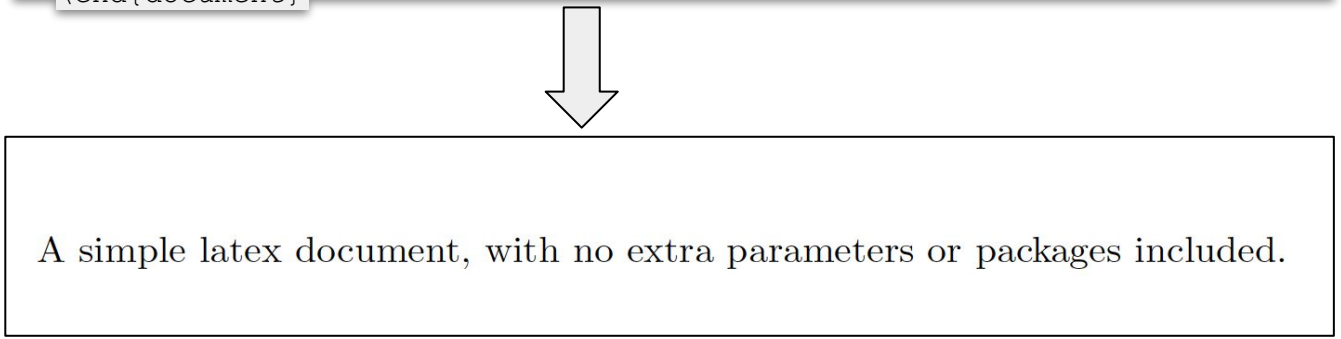
- **Procedural markup:** include instructions in the document that a program can interpret to print/display a document (e.g. Markdown).



Types of markup

- **Descriptive markup:** related to logical structure of a document, not necessarily related to presentation/display.

```
\documentclass{article}
\begin{document}
A simple latex document, with no  extra parameters or packages
included.
\end{document}
```



SGML (Standard Generalized Markup Language)

- A standard for defining generalized markup languages for documents.
- Uses a tree structure
- Elements have start/closing tags and content
- They can also have attribute name/value pairs in their tags
- Document type definition (DTD) define valid elements and attributes for an SGML application. Helps standardise application/enables validation of documents.

Example SGML DTD and document

```
<!ELEMENT anthology      - -  (poem+)>
<!ELEMENT poem           - -  (title?, stanza+)>
<!ELEMENT title          - O  (#PCDATA) >
<!ELEMENT stanza         - O  (line+)   >
<!ELEMENT line           - -  (#PCDATA) >
<!ATTLIST poem    id    ID   #IMPLIED status (draft|revised|published) draft   >

<anthology>
       <poem id=p1 status=revised>
         <stanza>
              <line>The first stanza.</line>
         <stanza>
              <line>Of a sad song</line>
              <line>this was.</line>
      </poem>
</anthology>
```

Hypertext Markup Language (HTML)

- An SGML application
- Domain specific language (DSL) for web markup
- Fixed tag set
- Tolerant to errors (capitalisation, missing start/end tags)
- Tags represent appearance and structure

```
<!DOCTYPE html>
<html lang="en">

    <head>
        <meta charset="UTF-8">
        <title>Hello!</title>
    </head>

    <body>
        <h1>Hello World!</h1>
        <p>This is a simple paragraph.</p>
    </body>

</html>
```

Hello World!

This is a simple paragraph.

eXtensible Markup Language (XML)

- Need to standardise HTML (browser war in the late 90s)
- Syntax (Strict notion of well-formedness)
- Namespaces
- Schemas in addition to DTDs
- eXtensible Stylesheet Language Transformations (XSLT). Uses XPath

XML Syntax

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE people_list [
  <!ELEMENT people_list (person*)>
  <!ELEMENT person (name, gender?)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT gender (#PCDATA)>
  <!ELEMENT IDnumber (#PCDATA)>]>

<people_list>
<person>
    <name>Fred Bloggs</name>
    <gender>Male</gender>
  </person>
</people_list>
```

Single root element, tags must be well-nested, cannot overlap, attribute values inside quotes.

Key characteristics of XML files

- Case-sensitive
- Spaces not allowed in names
- Element names must start with a letter or underscore
- Colons used to separate namespace in an element name

Encode reserved characters

The following characters are reserved and must be replaced by the corresponding named entity so that the project file can be parsed.

Reserved	Name entity
<	<
&	&
]]>	]]&
>	>
“	"
‘	'

Parsing XML

- **DOM (document object model) parsers**

  Random access to elements within document using DOM API

- **SAX parsers**

  Event-based model, which does not build complete document representation in memory. Suitable for large documents.

XML for configuration, data exchange and persistence

- Configuring Java project/build files using Maven's pom.xml (more on Maven in the next lecture)
- Application configuration
- REST API responses/requests
- Underlying format for data serialisation frameworks

Markdown

A lightweight markup language for creating formatted text using a plain-text editor.

```
Heading
=======
## Sub-heading
Paragraphs are separated by a blank line.
Text attributes _italic_, **bold**, `monospace`.
Horizontal rule: ---
Bullet list:
  * apples
  * oranges
  * pears
Numbered list:
  1. Waste
  2. Rinse
  3. Repeat
An [example](http://example.com)
![Image](Icon-pictures.png "icon")
```

Modern data languages

YAML Ain’t a Markup Language (YAML)

JavaScript Object Notation (JSON)

JSON Types

- strings
- numbers
- objects
- arrays
- Booleans (true or false)
- null

JSON syntax

```
{

"name": "Sarah",

"age" : 22,

"course": "SE",

"years": [2017, 2019],

"enrolled": true

}
```

Parsing JSON

In Java, you can use JSON-java library

```
import org.json.*;

String str = ... ; //assign your JSON String here i.e., "{ \"name\": \"Sarah\", \"age\": 22,
\"Course\": \"SE\" }"

JSONObject obj = new JSONObject(str);

String name = obj.getString("name");
int age = obj.getInt("age");
String course = obj.getString("Course");
```

YAML

- Superset of JSON
- Focus on human-readability and hand-editing
- Comments, multiline strings
- JSON style syntax or Python-like indented syntax
- Multiple documents in single file

YAML Types

- Numbers
- Strings
- Maps
- Lists
- Custom types using tags

YAML Example

```yaml
name: Rick
items:
  - item1
  - item2
```

As JSON

```json
{ "name":"Rick",

  "items": ["item1","item2"]
}
```

YAML aliases

Sharing data within a YAML document.

& labels an element. * references the label.

```
- &var item1
- *var
```

as JSON

```
["item1", "item1"]
```

JSON

```
{
  "name": "Sarah",
  "age": 22,
  "Course": "SE",
  "years": [2017,2019],
  "enrolled": true
}
```

YAML

```
name: Sarah
age: 22
Course: SE
years:
- 2017
- 2019
enrolled: true
```

YAML custom types

You can also create custom types (objects)

```
- !!Person {id: 42, name: Jim}
```