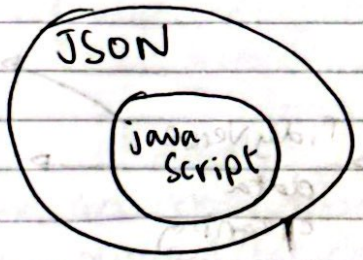
Webtech

JSON Data

- JSON - JavaScript object Notation
  - ↳format for representing data
  - ↳text based way to store and transmit structured data

- JSON is similar to XML (covered later)
  - ↳very widely used
  - lightweight and compact to send back and forth due to small site file
  - easy for both people and computers to read and write (much cleaner than XML)
  - maps very easily onto data structures
  -



→ Nicer integration (No syntax confusion)

- every major language has some way to parse

Config files

JSON Syntax

- Data Types:
  - string (double quotes " ")
  - number (decimals (rational numbers), integers)
  - boolean (true, false)
  - null

- Arrays:
  - like R vectors
  - sets of data types

- Objects
  - you can represent values as key-value pairs:
    {"key" : "value"}

You can use a combination of objects, arrays, and data types for more complex objects

Data Containers:

Ex:

```
{
  "name": ["X", "Y", "Z"],
  "grams": [300, 200, 500],
  "qty": [4, 5, null],
  "new": [true, false, true]
}
```

- these combinations allow for multiple representations for data

XML

- eXtensible Markup Language
- set of rules → encoding information
- used to describe data
  - ↳ semantic, hierarchical representation of data

Markup - a sequence of characters or other symbols inserted @ certain places in a document
↳ how content should be displayed
↳ describe documents structure

Marks in XML

- aka "tags" using <>

Ex:

<mark_name> Text </mark_name>

*NOTE:
- programming language
- network transfer protocol
- database

} XML IS NOT THIS

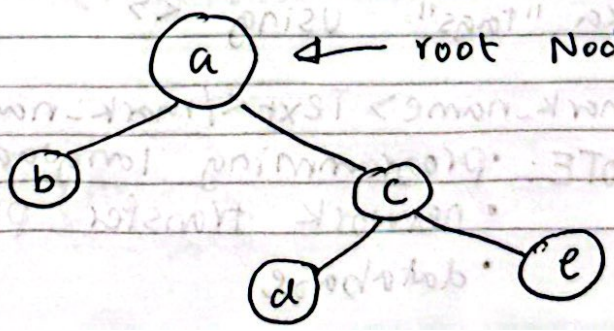
Some XML Dialects:

- KML (Keyhole Markup Language)
  - ↳ geo-spatial info (Google Earth, Google Sky)
- SVG (Scalable Vector Graphics)
  - ↳ graphical displays of 2D graphics w/ support for interactivity and animation
- PMML (Predictive Model Markup Language)
  - ↳ data mining & ML algorithms/models
- RSS (Rich Site Summary)
  - ↳ feeds for publishing blog entries
- SDMX (Statistical Data & Metadata Exchange)
  - ↳ organizing & exchanging info
- SBML (Systems Biology Markup Language)
  - ↳ bio systems

Ex: Good Will Hunting

```
<movie mins="126" lang="en">
    <title> Good Will Hunting </title>
    <director> Gus Van Sant </director>
    <year> 1998 </year>
    <genre> drama </genre>
</movie>
```

* this type of structure allows us to make "Tree" data structure



- XML is a "well-formed" language

What is a well-formed document?
- one root element containing rest of elements
- properly nested elements
- self-closing tags
- attributes appear in start-tags of elements
- attribute values must be quoted
- element names & attribute names are case sensitive

*IF XML documents are not well formed

Additional XML Elements

<?xml>	XML Declaration	• Identifies content as XML Document
<?PI>	Processing Instruction	• Processing instructions passed to application PI
<!DOCTYPE>	Document-type Declaration	• Defines structure of an XML document
<![CDATA[ ]]>	CDATA Character Data	• Anything inside CDATA is ignored by parser
<!-- -->	Comment	

Example

```
<?xml version="1.0"?>
<!DOCTYPE movies>
<movie mins="126" lang="en">
  <!-- Comment -->
  <title>Good Will Hunting</title>
  <director>
    <first_name>Gus</first_name>
    <last_name>Van Sant</last_name>
  </director>
  <year>1998</year>
  <genre>drama</genre>
</movie>
```

Parsing XML

- R libraries:

```
library(xml2)
library(stringr)
```

Parsing

- analyze (a sentence or phrase) into parts and describing the syntax

4 Major Tasks (xml2):

- parsing xml/html content
- obtaining descriptive info about parsed contents
- navigating the tree structure (accessing parts of tree)
- querying and extracting data from parsed contents

Parsing Functions

- read_xml()
- read_html()

Read XML (read_xml):

- use for any and all xml files (assuming they are well formed)
  - ↳ "go to" function to use

Read HTML (read_html):

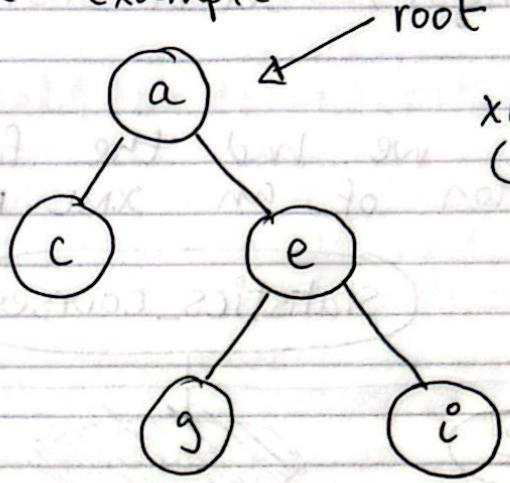
- more robust
  - ↳ can handle non well-formed files

Input: String, R Connection, Vector
Output: XML Document

What is an "XML Document"
↳ basically a list of XML Tags

- Once we parse, we have 2 functions to access elements of the tree:
  - xml_root() → returns root
  - xml_children() → children of a given node

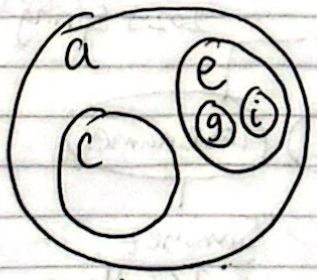
Conceptual Example:



xml_root (xml Document)
↳ a { c (No children)
e
↳ g } No children
↳ i }

xml_children (XML doc)
↳ c, e{g, i}

* Remember our tree is a nested structure, so it looks like this:

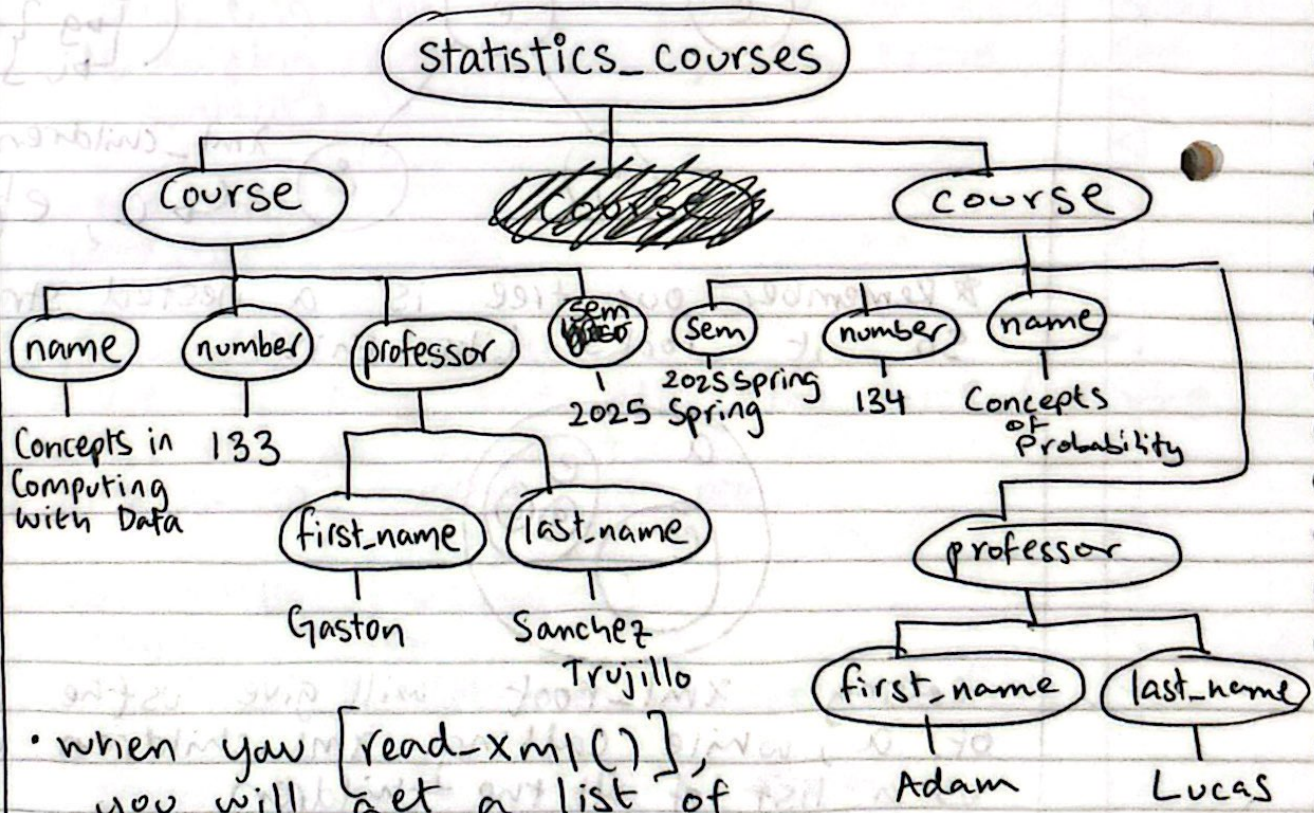


calling xml_root will give us the contents of a, while calling xml_children gives us a list of all the children

Some more xml navigation functions:
- xml_root() → return root node
- xml_children() → returns children nodes
- xml_child() → return specific child

xml_name() → name node
xml_contents() → return contents of a node
xml_text() → text
xml_length() → number of children nodes
xml_parents() → returns set of parent nodes
xml_siblings() → set of sibling nodes

Example:
Suppose we had the following tree representation of an XML Document:



- when you [read_xml()], you will get a list of

\n ↳ indicates new line

```
#> {xml_document}
#> <statistics_courses>
#> [1] <course>\n<name>Concepts in ...
#> [2] <course>\n<name>Concepts of Probability...
```

Interestingly, the XML root and now we parsed and returned XML document are identical

```
a = read_xml(...)  → XML Document

root_a = xml_root(a)

identical(a, root_a) → TRUE
```

this because of the nesting structure, our root contains all the children, hence the entire XML Document!

```
xml_length(a) → 2  (# of courses (number of nodes in root node))

xml_child(a, search=1) → get's the first child or Stat 133 course in this case
```

We can also search by tag:

```
Stat133 = xml_child(a, search = 1)

prof = xml_child(stat133, "professor")

xml_text(prof)

#> [1] Gaston Sanchez Trujillo
```

XPath

- We can parse documents (xml library)
  - ↳ even further:
    locate nodes & extract info (xml_text(...))

What is XPath?
- language to find info about an XML doc
- identifying patterns to match data and content (like Regex expressions : str_match())

Ex:
/movies/movie

Selecting Nodes

- main symbols to define path expressions:

/	→	selects from root
//	→	nodes anywhere
.	→	current node
..	→	parent of current node
@	→	attributes
[]	→	indicate attributes
*	→	matches element node
@*	→	matches attribute node

XPath Examples

- Let's take the Statistics courses tree from the XML section

Ex 1: /statistics_courses/course

- Gets the 2 course nodes

Ex 2: /statistics_courses/course/name

- Both "name" nodes

Ex 3: /statistics_courses/*

↳ same as:

/statistics_courses/course

- Both expressions get both course nodes

Ex 4: //first_name

- first_name nodes anywhere in the XML tree

How can we specifically get one course?

- IF we use attributes we can get a specific path

Ex: course order='1' course order='2'

/statistics_courses/course[@order='1']

↳ gets the left side of the tree (statistics_courses)

Let's use the statistics_courses tree:

a = read_xml(...) → returns xml document

number_nodes = xml_find_all(a, xpath= //course/number)

xml_text(number_nodes)

↳ #>[1] 133 134

Introduction

HTML

- not a programming language → markup language
  - ↳ like XML and JSON
- you can take any website → Right Click → View Source / Developer / Inspect

You get a HTML written portion of the website

HTML doc structure

<!DOCTYPE html> ← HTML Doctype declaration

```
<html lang="en">
    <head>
        ...
    </head>
    <body>
        ...
    </body>
</html>
```

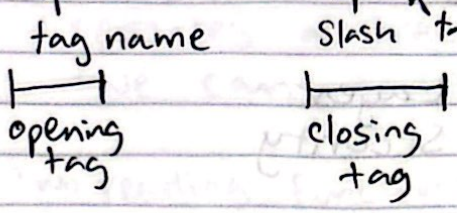
(<html lang="en"> ... </html> = html element; <head> ... </head> = head element; <body> ... </body> = body element)

head - descriptive info such as title, style, scripts, etc

body - everything that is to be displayed

HTML Syntax

 content



* You don't always need a pair of matching tags (some don't have closing tags)

Ex: <img>

closes itself

Attributes:

Que tal!

attribute name — attribute value

- Specify additional info about element

Browser

- any major browser reads HTML, interprets the tags, then renders

<h1>, <h2> : headings
 : paragraph

- CSS (Cascade Style Sheets) → helps for other visual aspects of a page

Web Scraping

Websites → SSL/TSL

- SSL - Secure Sockets Layer
- TLS - Transport Layer Security
  - ↳ keeps an internet connection secure and safeguards sensitive data

HTTPS → secure extension of HTTP

Ex:

```
wiki <- 'https://en.wikipedia.org/...

tbls <- readHTMLTable(wiki) → fails
```

So one way to read them is by downloading the html file then read locally

Summary

Webtech

- JSON
- XML

} use HTML syntax

- HTML - mark up language (NOT PROGRAMMING)

JSON:

- super set of Java script
- Data Types: string, boolean, number, null
- Arrays
- Objects

XML:

- sets of marks <tag> ... </tag>
- creates a tree structure organization
- we can parse XML with the library(xml2)

Two reading functions: (in library(xml2))

read_xml()
read_html()

XPath:

- using a path syntax to locate nodes in XML documents

Web Scraping:

- HTTPS ← secured HTTP
  - ↳ download HTML files
  - read HTMLTable(...)
    - ↳ traverse HTML according and you can perform data analysis!

THANK YOU for a wonderful semester!