XML, XPath & R

Presentation Structure

XML - really basic overview

Xpath - a query language for XML

Implementation in R

XML - Overview

Language for describing data

Contains data and metadata *(data about data)*

*This means XML documents contain **both** content and markup*

Hierarchical Structure

*Inverted tree structure with a **single** root*

XML - Overview

Written in plain text *(you can use notepad)*

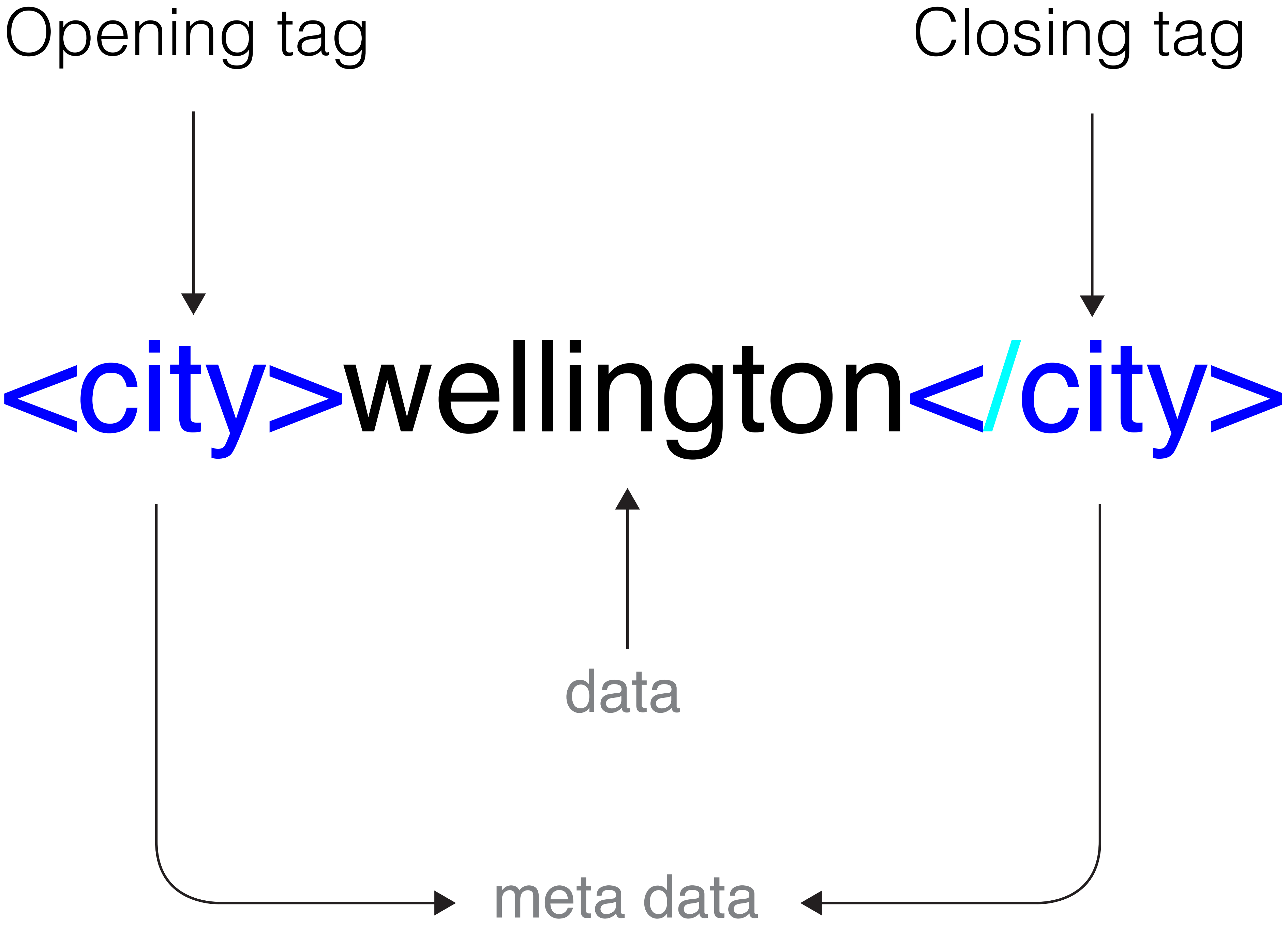
Examples of XML: *(HTML = Hypertext Markup Language; SVG = Scalable Vector Graphics; PMML = Predictive Modelling Markup Language; SDMX; RDF)*

Many many others - https://en.wikipedia.org/wiki/List_of_XML_markup_languages

Well formed Vs Valid

Well formed = consistent with XML syntax
Valid = consistent with schema semantics

An XML element



Opening tag

Closing tag

<city>wellington</city>

data

meta data

XML attributes

```
<city hemisphere = “south”>sydney</city>
```

Attributes add additional information to the element.

Matter of judgement whether to model information as an attribute or as an element.

Many elements (style A)

```
<cities>
    <city hemisphere = "south">sydney</city>
    <city hemisphere = "south">wellington</city>
    <city hemisphere = "north">paris</city>
    <city hemisphere = "north">beijing</city>
</cities>
```

Many elements (style B)

```
<cities>
    <north_hemisphere>
        <city>paris</city>
        <city>beijing</city>
    <north_hemisphere>

    <south_hemisphere>
        <city>sydney</city>
        <city>wellington</city>
    <south_hemisphere>
</cities>
```

Worked example

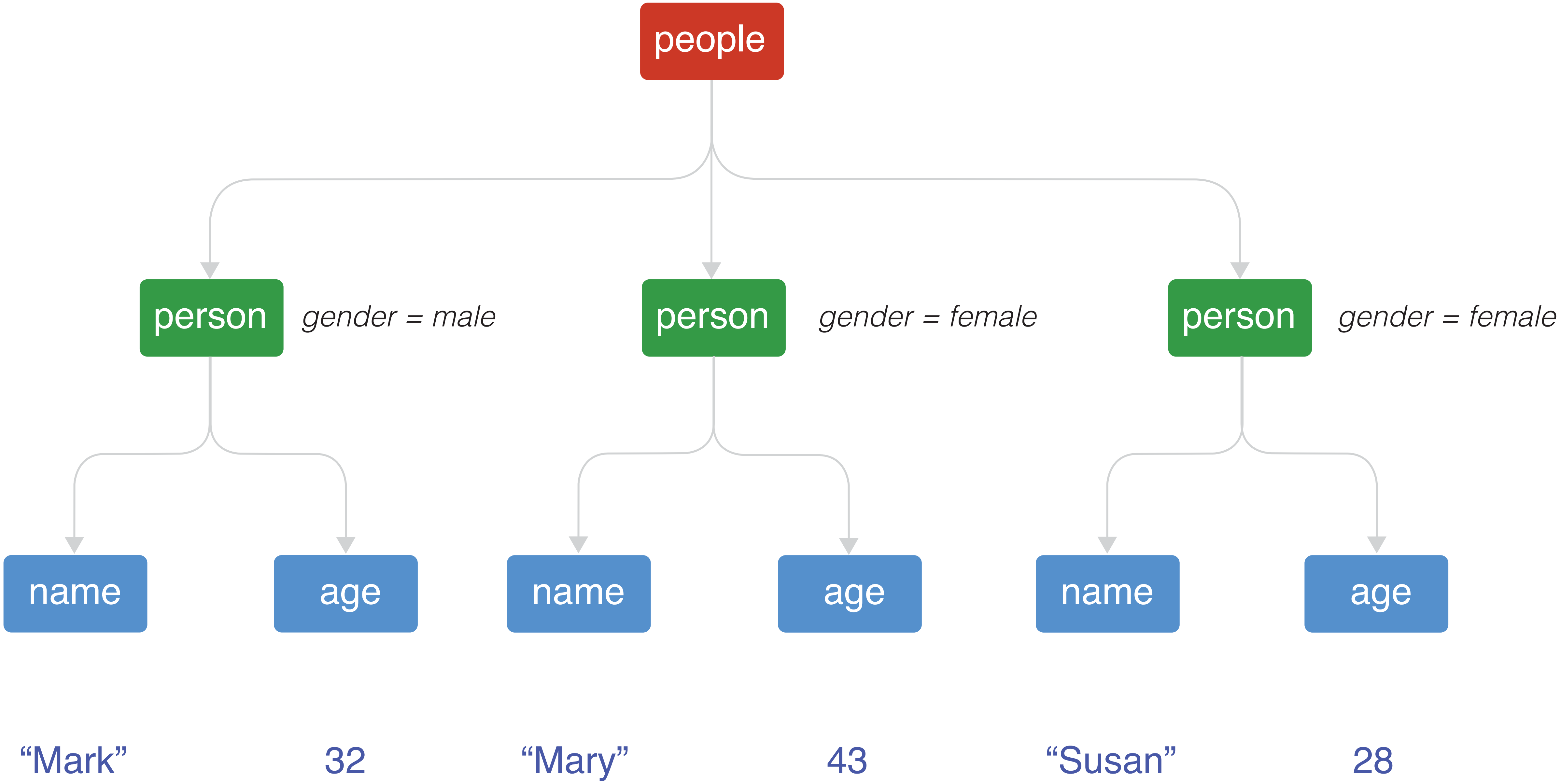
An example dataset of three people

Name	Age	Gender
Mark	32	Male
Mary	43	Female
Susan	28	Female

The above example shows a rectangular block of data.

XML can represent much more complex structures than shown above.

Inverted Tree Structure



Translated into XML

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
        <person gender = "male">
                <name>Mark</name>
                <age>32</age>
        </person>
        <person gender = "female">
                <name>Mary</name>
                <age>43</age>
        </person>
        <person gender = "female">
                <name>Susan</name>
                <age>28</age>
        </person>
</people>
```

JSON Equivalent

```
{“people":{"person":

[{"gender":"male","name":"Mark","age":32},

{"gender":"female","name":"Mary","age":43},

{"gender":"female","name":"Susan","age":28}]}}
```

JSON is more terse than XML as XML includes the “angle bracket tax”

The smaller size is why JSON may be preferred in web services.

XML Technologies

Xpath	Extracting information - fairly simple.
XQuery	Multiple Xpath statements - comprehensive.
Xinclude	Composing complex documents
Xlink	Links within / between XML docs.
XPointer	Links to specific parts of documents (Xpath)
XSLT	Transforms documents. XML *to* CSV / HTML
XML - FO	Transforms XML into PDF / Epub.
XML - Schema	Schema language / specifies XML documents.

Xpath expression (1)

“/people/person/name”

Select all the name elements under the people/person element

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
        <person gender = "male">
                <name>Mark</name>
                <age>32</age>
        </person>
        <person gender = "female">
                <name>Mary</name>
                <age>43</age>
        </person>
        <person gender = "female">
                <name>Susan</name>
                <age>28</age>
        </person>
</people>
```

Xpath expression (2)

“//name”

Select all the name elments within the document (regardless of where they appear)

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
        <person gender = "male">
                <name>Mark</name>
                <age>32</age>
        </person>
        <person gender = "female">
                <name>Mary</name>
                <age>43</age>
        </person>
        <person gender = "female">
                <name>Susan</name>
                <age>28</age>
        </person>
</people>
```

Xpath expression (3)

“/people/person[2]/name”

Select all the name elements from the second person.

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
        <person gender = "male">
                <name>Mark</name>
                <age>32</age>
        </person>
        <person gender = "female">
                <name>Mary</name>
                <age>43</age>
        </person>
        <person gender = "female">
                <name>Susan</name>
                <age>28</age>
        </person>
</people>
```

Xpath expression (4)

“/people/person[age < 35]”

Select all the person elements where the age of the person is less than 35

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
        <person gender = "male">
                <name>Mark</name>
                <age>32</age>
        </person>
        <person gender = "female">
                <name>Mary</name>
                <age>43</age>
        </person>
        <person gender = "female">
                <name>Susan</name>
                <age>28</age>
        </person>
</people>
```

Xpath expression (5)

“/people/person[age < 35]/name”

*Select all the **name** elements where the age of the person is less than 35*

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
        <person gender = "male">
                <name>Mark</name>
                <age>32</age>
        </person>
        <person gender = "female">
                <name>Mary</name>
                <age>43</age>
        </person>
        <person gender = "female">
                <name>Susan</name>
                <age>28</age>
        </person>
</people>
```

Xpath expression (6)

“/people/person[@gender = 'female']/age”

Find out the ages of all women

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
        <person gender = "male">
                <name>Mark</name>
                <age>32</age>
        </person>
        <person gender = "female">
                <name>Mary</name>
                <age>43</age>
        </person>
        <person gender = "female">
                <name>Susan</name>
                <age>28</age>
        </person>
</people>
```

Xpath expression (7)

“/people/person/@gender”

List the gender attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
        <person gender = "male">
                <name>Mark</name>
                <age>32</age>
        </person>
        <person gender = "female">
                <name>Mary</name>
                <age>43</age>
        </person>
        <person gender = "female">
                <name>Susan</name>
                <age>28</age>
        </person>
</people>
```

Xpath expression (8)

`/people/person[contains(name, 'Mar')]/@gender`

List the genders for names which contain the phrase “Mar”

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
        <person gender = "male">
                <name>Mark</name>
                <age>32</age>
        </person>
        <person gender = "female">
                <name>Mary</name>
                <age>43</age>
        </person>
        <person gender = "female">
                <name>Susan</name>
                <age>28</age>
        </person>
</people>
```

Xpath expression (9)

Equivalent statements

```
/people/person[1]/following-sibling::person[1]/name
/people/person[3]/preceding-sibling::person[1]/name
```

“Following-sibling” is called an Xpath “axes” there are 13 of these.

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
        <person gender = "male">
                <name>Mark</name>
                <age>32</age>
        </person>
        <person gender = "female">
                <name>Mary</name>
                <age>43</age>
        </person>
        <person gender = "female">
                <name>Susan</name>
                <age>28</age>
        </person>
</people>
```

Xpath expression (10)

```
count(/people/person)
```

The above returns 3

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
        <person gender = "male">
                <name>Mark</name>
                <age>32</age>
        </person>
        <person gender = "female">
                <name>Mary</name>
                <age>43</age>
        </person>
        <person gender = "female">
                <name>Susan</name>
                <age>28</age>
        </person>
</people>
```

Tree Terminology

Siblings - same level in the tree

Children - immediate level down.

Parent - immediate level up.

Dependents - all levels down.

Ancestors - all levels up.

XML in R

Two main libraries: XML & xml2 (Wickham)

Both use Xpath 1.0 not 2.0

XML is more comprehensive than xml2

XML in R (1)

Load an XML document

```
#load in library
library(XML)

#get the xml document from disk
doc_xml <- XML::xmlParseDoc(“xml_people.xml”)

class(doc_xml)
[1] "XMLInternalDocument"
```

XML in R (2)

Run an Xpath expression and extract elements

```
> result <- xpathSApply(doc_xml, “/people/person/name")

> class(result)
[1] “list"

> result
[[1]]
<name>Mark</name>
[[2]]
<name>Mary</name>
[[3]]
<name>Susan</name>
```

XML in R (2)

Run an Xpath expression and extract text

```
> result <- xpathSApply(doc_xml, "/people/person/name",
xmlValue)
> class(result)
[1] “character"

> result
[1] "Mark"  "Mary"  "Susan"
```

XML in R (3)

Run an Xpath expression using attribute condition

```
> result <- xpathSApply(doc_xml, “/people/person[@gender
                 = 'female']")
> result
```

```
[[1]]
<person gender="female">
        <name>Mary</name>
        <age>43</age>
    </person>

[[2]]
<person gender="female">
        <name>Susan</name>
        <age>28</age>
    </person>
```

XML in R (4)

Run an Xpath expression and extract attribute value

```
> result <- xpathSApply(doc_xml, “/people/person[@gender
              = ‘female’]", xmlGetAttr, "gender")
> result
```

```
[1] "female" "female"
```

XML in R (5)

Create a simple node

```
> n1_name <- xmlNode("name", "Mark")

> class(n1_name)
[1] "XMLNode"          "RXMLAbstractNode" "XMLAbstractNode"

> n1_name
<name>Mark</name>
```

node creation

XML in R (6)

Create a blank node with an attribute

```
> n1_person <- xmlNode("person", attrs = c(gender =
            “male"))

> n1_person

<person gender="male"/>
```

node creation

XML in R (7) *Add a list of children*

```
> lst_children_1 <- list(
+                          xmlNode("name", "Mark"),
+                          xmlNode("age", 32)
+                  )
>
> n1 <- addChildren(n1_person, kids = lst_children_1)
> n1
<person gender="male">
 <name>Mark</name>
 <age>32</age>
</person>
```

node creation

XML in R (8) *Join the bits together*

```
> node_root <- xmlNode("people")
> xml_document <- addChildren(node_root, kids = list(n1,
                   n2, n3))
> xml_document
```

```
<people>
 <person gender="male">
  <name>Mark</name>
  <age>32</age>
 </person>
 <person gender="female">
  <name>Mary</name>
  <age>43</age>
 </person>
 <person gender="female">
  <name>Susan</name>
  <age>28</age>
 </person>
</people>
```

XML in R (9)

Save the document

```
XML::saveXML(xml_document, "xml_people.xml")
```

Retrieving SMH Headlines

```r
library(httr)
library(xml2)
library(dplyr)

str_url <- "smh.com.au"

# send an http request and get current smh.com.au page
# results are stored as a string
str_c_page <- httr::content(httr::GET(str_url), as = 'text')

# parse the looonnnggg string into xml object
html_c_page <- xml2::read_html(str_c_page)

# lazy way of plucking out all h2 elements
lst_node_set <- xml2::xml_find_all(html_c_page, "//h2")

# iterate through the list and get the text contents
vct_h2_headlines <- lst_node_set %>% sapply(xml_text)

# print to screen
vct_h2_headlines
```