Scraping using XPath

Max Turgeon

SCI 2000–Introduction to Data Science

Lecture Objectives

- Scrape data using XPath and understand its basic components
- Compare and contrast CSS selectors and XPath

Motivation

- Last week, we discussed how to extract data from HTML files.
- We focused on CSS selectors, which is a way to identify which element we want.
- However, we also saw that sometimes it extracts more information than we need.
    - E.g. Tables from Wikipedia all have the same class.
- **XPath** is a powerful and more precise way of describing a specific element inside an HTML file
    - E.g. It also applies to XML files more generally.

Example i

```
# From last week----
library(rvest)
library(tidyverse)

url <- "https://en.wikipedia.org/wiki/World_population"
world_pop_tables <- read_html(url) %>%
  html_elements("table.wikitable")

length(world_pop_tables)
```

```
## [1] 13
```

Example ii

```
# Notice the single and double quotes!
xpath <- '//div[@id="mw-content-text"]/div[1]/table[4]'
exact_table <- read_html(url) %>%
  html_element(xpath = xpath)

library(knitr)
html_table(exact_table) %>%
  select(Rank, Country, Population) %>%
  kable()
```

Example iii

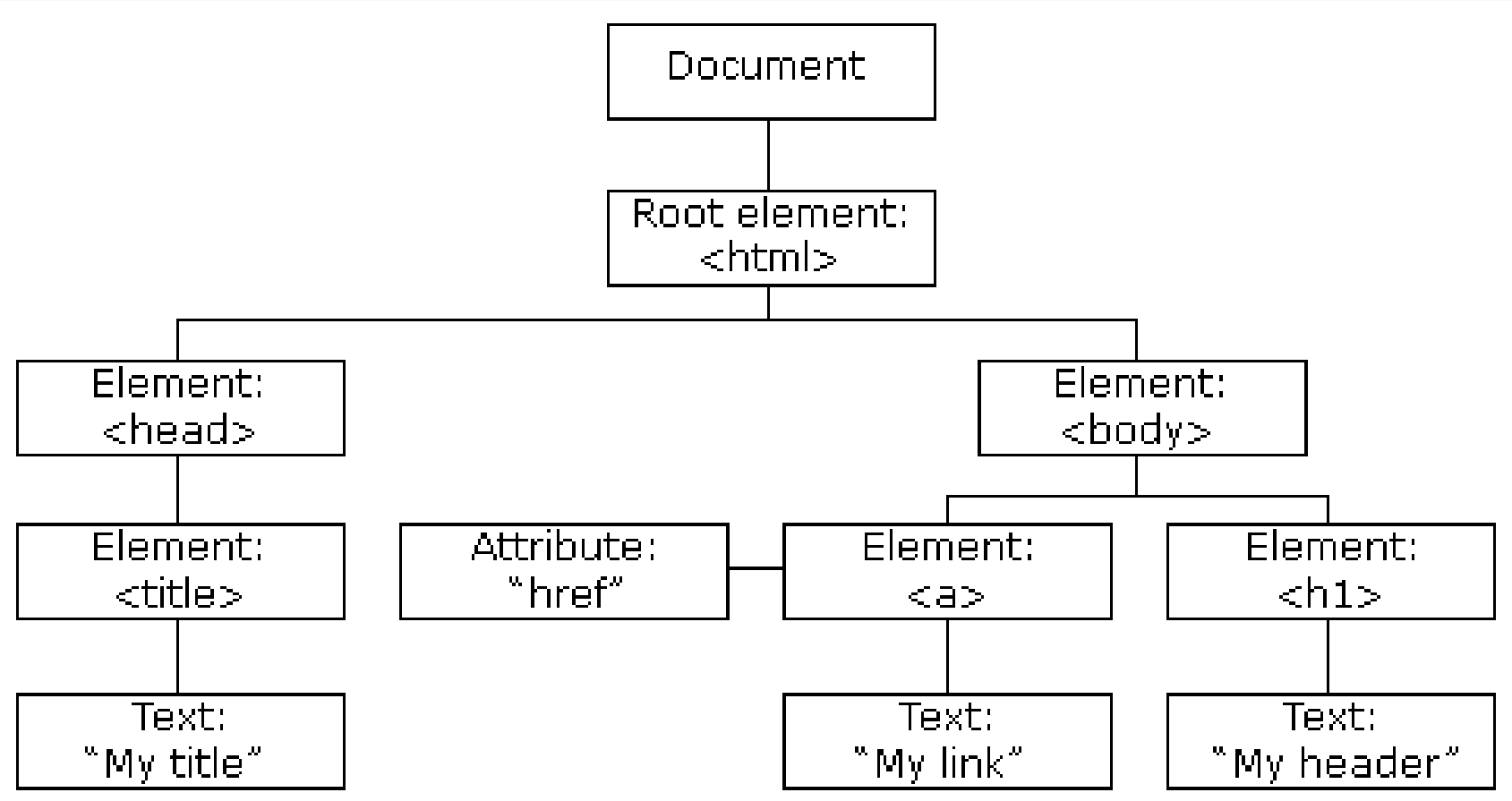
Rank	Country	Population
1	China	1,407,257,360
2	India	1,375,030,248
3	United States	331,402,425
4	Indonesia	269,603,400
5	Pakistan	220,892,331
6	Brazil	212,925,393
7	Nigeria	206,139,587
8	Bangladesh	170,397,280
9	Russia	146,748,590
10	Mexico	127,792,286

XPath—Basic syntax i

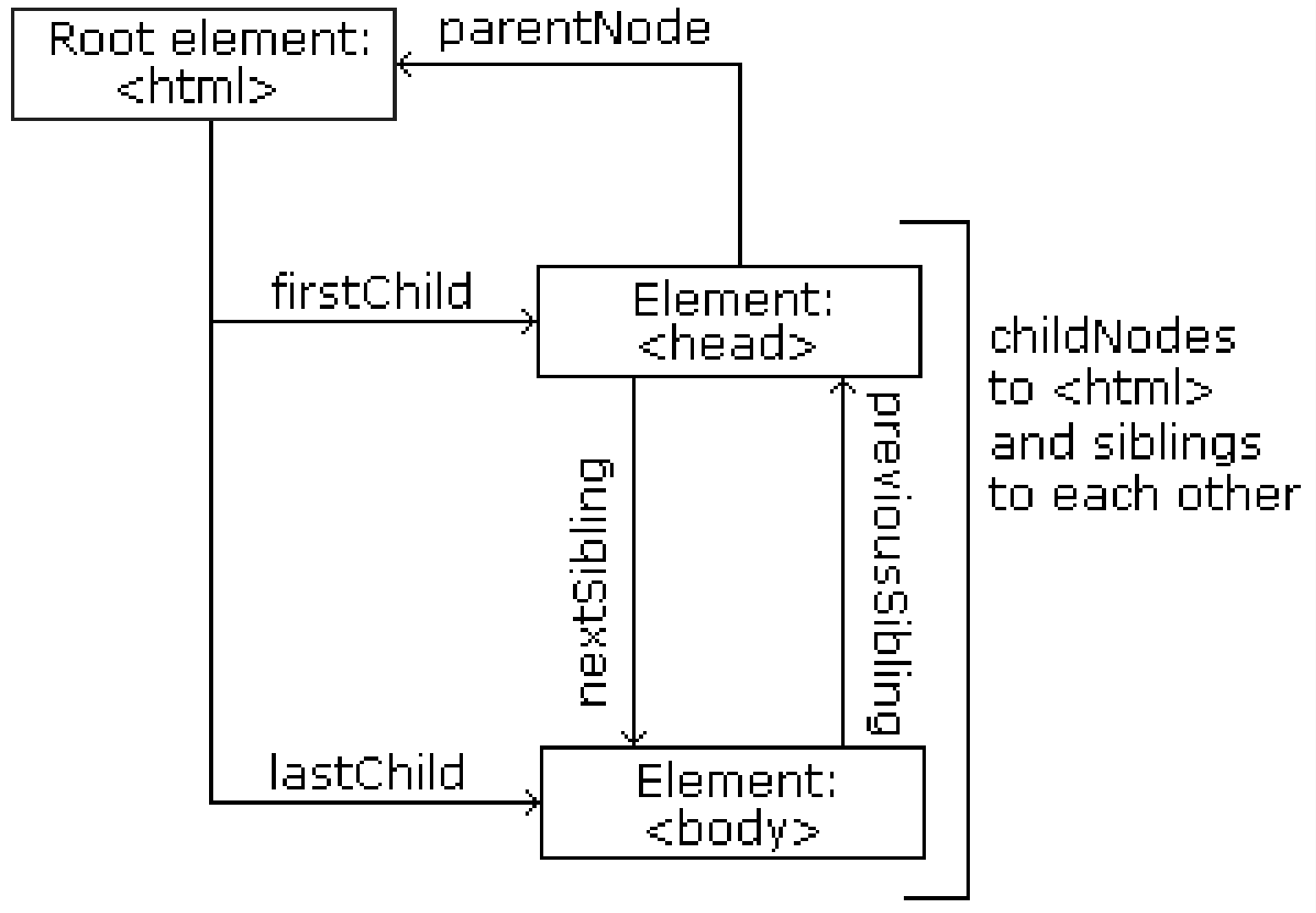
- **XPath** uses path expressions (think “file system”) to select elements in an HTML document.
- We can specify paths:
  - based on element names (e.g. `div` or `p`)
  - based on element attributes (e.g. `class` or `href`)
  - based on an element’s relationship to other elements (e.g. `p` inside `div`)
- Let’s look at the path we used earlier
  - `//div[@id="mw-content-text"]/div[1]/table[4]`
- You probably recognize some of the elements here (`div` and `table`).

XPath—Basic syntax ii

- But what do the other pieces mean?



XPath—Basic syntax iii



XPath—Basic syntax iv

- In other words, XPath cares about:
  - The type of element and its attributes (just like CSS selectors)
  - The ancestor/descendant relationship between elements (in a more refined way than CSS selectors)
  - The child order within a generation (this is new!)
- With all of this, we can create very specific search strings in a way CSS selectors simply can't.

XPath—Basic syntax ∨

- Going back to our XPath example:
  `//div[@id="mw-content-text"]/div[1]/table[4]`
  - `div[@id="mw-content-text"]` matches a `div` with a specific `id`.
  - `div[1]` matches the first `div` child of the previous element.
  - `table[4]` matches the fourth `table` child of the previous element.
  - The starting `//` means this match could occur anywhere in the HTML document.

Exercise

On `https://www.r-project.org/mail.html`, you can find a list of *Special Interest Group* mailing lists. Create an XPath that will match all `a` elements from this list, and no other ones.

Hint: Open up the developer tools, start from `/html/body` and go from there.

Solution i

```r
library(rvest)

url <- "https://www.r-project.org/mail.html"
path <- "/html/body/div/div[1]/div[2]/ul[1]/li/p/a"

mail <- read_html(url) %>%
    html_elements(xpath = path)
```

Solution ii

```
library(tidyverse)

data.frame(
    name = html_text(mail),
    URL = html_attr(mail, "href")
) %>% glimpse
```

```
## Rows: 20
## Columns: 2
## $ name <chr> "R-SIG-Mac", "R-SIG-DB",
"R-SIG-Debian", "R-SIG-dynamic-models", ~
```

Solution iii

```
## $ URL <chr>
”https://stat.ethz.ch/mailman/listinfo/r-sig-mac”,
”https://stat.~
```

Solution iv

```
# Equivalently: once we reach ul[1]
# we want all a elements
path2 <- "/html/body/div/div[1]/div[2]/ul[1]//a"

mail2 <- read_html(url) %>%
    html_elements(xpath = path)

# Are they the same?
all.equal(mail, mail2)
```

```
## [1] TRUE
```

Axis specifiers

Expression	Description
`@`	Extract an attribute
`.`	Refers to current node
`..`	Refers to (direct) parent node
`\\`	Refers to direct or indirect children

Example i

- On `https://coinmarketcap.com/all/views/all/`, there is a table with information about crypto-currencies.
- Let's extract this data and find the top 5 crypto-currencies with respect to their price (in USD).
- We can see our `table` of interest is inside a `div` of class `cmc-table__table-wrapper-outer`

Example ii

```
library(tidyverse)
url <- "https://coinmarketcap.com/all/views/all/"
path <- paste0('//div[@class="cmc-table__',
               'table-wrapper-outer"]/div/table')

data <- read_html(url) %>%
    html_element(xpath = path) %>%
    html_table()
glimpse(data)
```

Example iii

```
## Rows: 200
## Columns: 11
## $ Rank <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15~
## $ Name <chr> "Bitcoin", "Ethereum", "Binance
Coin", "Tether", ~
## $ Symbol <chr> "BTC", "ETH", "BNB", "USDT",
"ADA", "DOT", "XRP",~
## $ `Market Cap` <chr> "$1,072,271,071,091",
"$209,184,934,190", "$42,17~
## $ Price <chr> "$57,438.84", "$1,815.00",
"$272.92", "$0.9996", ~
```

Example iv

```
## $ `Circulating Supply` <chr> "18,668,050 BTC",
"115,253,326 ETH", "154,532,785~
## $ `Volume(24h)` <chr> "$56,767,684,476",
"$22,431,621,017", "$2,062,433~
## $ `% 1h` <chr> "-0.68%", "-0.84%", "-0.42%",
"0.01%", "-0.55%", ~
## $ `% 24h` <chr> "4.04%", "8.33%", "3.14%",
"-0.15%", "2.29%", "7.~
## $ `% 7d` <chr> "2.30%", "3.68%", "1.67%",
"-0.39%", "4.13%", "-7~
## $ `` <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA,
NA, NA, NA, N~
```

Example v

```
# Clean up Price so we can order it
# NOTE: there's a column name missing
# which can cause some weird errors
library(stringr)
data %>%
    select(Name, Symbol, Price) %>%
    mutate(Price = str_replace_all(Price, "\\$|,", ""),
           Price = as.numeric(Price)) %>%
    top_n(5, Price)
```

Example vi

```
## # A tibble: 5 x 3
##   Name             Symbol  Price
##   <chr>            <chr>   <dbl>
## 1 Bitcoin          BTC    57439.
## 2 Wrapped Bitcoin WBTC   57426.
## 3 Bitcoin BEP2     BTCB   57813.
## 4 yearn.finance    YFI    35399.
## 5 renBTC           RENBTC 57221.
```

Exercise

On `https://en.wikipedia.org/wiki/List_of_cognitive_biases`, you can find multiple tables about cognitive biases. Focusing on social biases, extract the link to the different biases (i.e. from the first column). Be careful not to extract the other links (you should have 42 links).

Bonus: Using these links, extract the list of references for each social cognitive bias.

Solution i

- First, we observe that our table of interest is the second `table` inside a `div`: `//div/table[2]`
- Next, we see that inside the table, each bias is organized into a row (`tr`), and that within a row, we want to focus on the first entry (`td[1]`). The `a` element we want is there, which gives us `tr/td[1]/a`.
- This second piece is an indirect child of the first piece, so we have our path: `//div/table[2]//tr/td[1]/a`

Solution ii

```
url <- paste0("https://en.wikipedia.org/wiki/",
              "List_of_cognitive_biases")
path <- "//div/table[2]//tr/td[1]/a"

list_links <- read_html(url) %>%
  html_elements(xpath = path) %>%
  html_attr("href")

str(list_links)
```

Solution iii

```
## chr [1:42] "/wiki/Actor-observer_bias"
"/wiki/Authority_bias" ...
```

- For the references, note that they are stored in an ordered list (`ol`) of class `references`, and each citation is inside a special `cite` element.
    - The XPath we need is `//ol[@class="references"]//cite`

Solution iv

```
# It's a good idea to test before looping
path_ref <- '//ol[@class="references"]//cite'

paste0("https://en.wikipedia.org/",
       list_links[1]) %>%
  read_html() %>%
  html_elements(xpath = path_ref) %>%
  html_text() %>%
  str()
```

Solution v

```
## chr [1:21] "Miller, Dale; Normal, Stephen
(1975). \"Actor-observer differences in
perceptions of effective control\". Journ"|
__truncated__ ...
```

Solution vi

```
# Write a function
extract_refs <- function(url) {
  paste0("https://en.wikipedia.org/",
         url) %>%
  read_html() %>%
  html_elements(xpath = path_ref) %>%
  html_text()
}
```

Solution vii

```
# Double check
extract_refs(list_links[1]) %>%
  str
```

```
## chr [1:21] "Miller, Dale; Normal, Stephen
(1975). \"Actor-observer differences in
perceptions of effective control\". Journ"|
__truncated__ ...
```

Solution viii

```
# Loop over all links----
library(purrr)

full_refs <- map(list_links,
                 extract_refs)

# full_refs is a list
length(full_refs)
```

```
## [1] 42
```

Summary

- XPath gives us more flexibility than CSS selectors by focusing on the relationship between elements.
- Some developer tools can give you an XPath for a specific element—look up online for more details!
- XPath is a lot more complex than we have time to discuss.
    - In fact, XPath is a Turing-complete (query) language.