Web Scraping with R (1): Parsing HTML

Alex Sanchez and Francesc Carmona
Genetics Microbiology and Statistics Department

Universitat de Barcelona
October 2022

Outline

1) Introduction: What is *parsing*

2) Parsing HTML with $rvest$`

3) Using CSS selectors to locate information

4) References and Resources

Introduction: What is *parsing*?

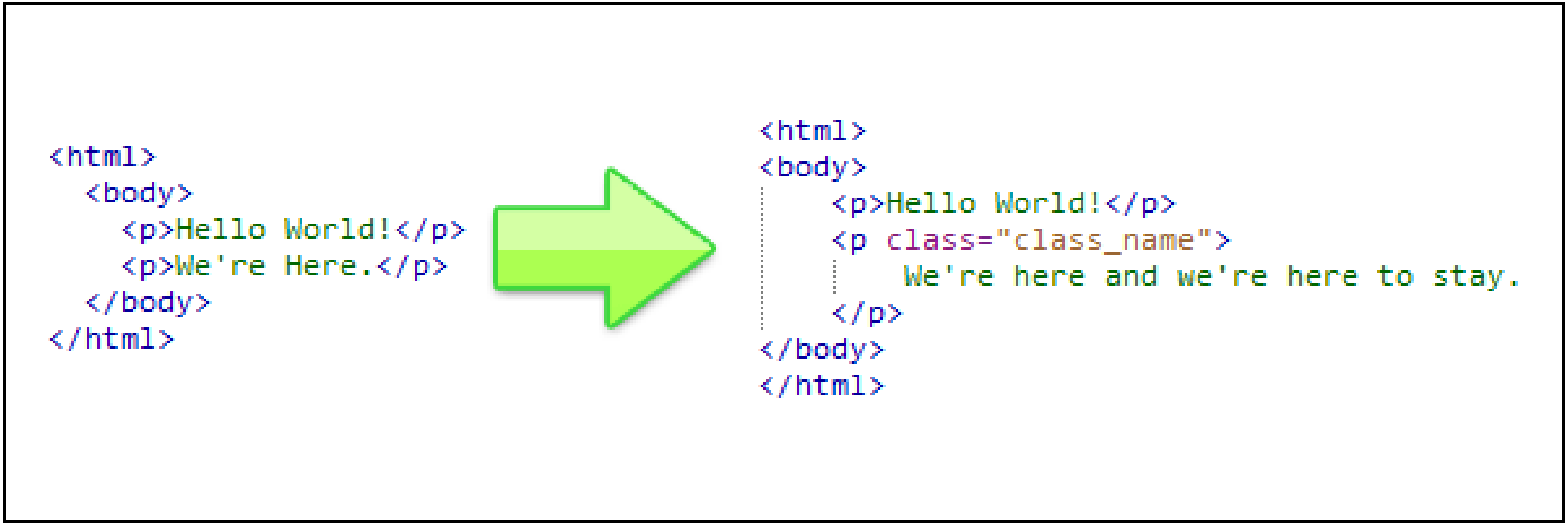
Introduction to parsing

- Scraping HTML pages usually done in two steps:
  - First, desired content from the Web is examined to determine if it is actionable to further analyses.
  - Second, HTML files are read and information is extracted from them.
- Parsing HTML occurs at both steps
  - *by the browser* to display HTML content nicely, and also
  - *by parsers in R* to construct useful representations of HTML documents in our programming environment.

What is *parsing*

Parsing involves *breaking down a text into its component parts of speech with an explanation of the form, function, and syntactic relationship of each part.* Wikipedia.

```r
knitr::include_graphics("images/parseHTML.png")
```



```html
<html>
  <body>
    <p>Hello World!</p>
    <p>We're Here.</p>
  </body>
</html>
```

```html
<html>
<body>
    <p>Hello World!</p>
    <p class="class_name">
        We're here and we're here to stay.
    </p>
</body>
</html>
```

Reading vs parsing

- Not just a semantic difference:
    - **reading** relies on functions that *do not care about the formal grammar that underlies HTML*, only recognizing the sequence of symbols included in the HTML file.
    - **parsing** employs programs that understand the special meaning of the mark-up structure reconstructing the HTML hierarchy within some R-specified structure.

Getting data (1): *Reading* an HTML file

- HTML files are text files, thus, they can be read using the `readlines()` function:

```
url <- "http://www.r-datacollection.com/materials/html/fort
fortunes <- readLines(con = url)
head(fortunes, n=10)
```

```
##  [1] "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML//EN\">"
##  [2] "<html> <head>"
##  [3] "<title>Collected R wisdoms</title>"
##  [4] "</head>"
##  [5] ""
##  [6] "<body>"
##  [7] "<div id=\"R Inventor\" lang=\"english\" date=\"June/200
##  [8] "  <h1>Robert Gentleman</h1>"
```

readLines() is a *reading* function

- maps every line of the input file to a separate value in a character vector creating a flat representation of the document.

- it is *agnostic* about the different tag elements (name, attribute, values, etc.),

- it produces results that do not reflect the document's internal hierarchy *as implied by the nested tags* in any sensible way.

Getting data (2): parsing an HTML file

- To achieve a useful representation of HTML files, we need to employ a program that:
  - understands the special meaning of the markup structures, and
  - reconstructs the implied hierarchy of an HTML file within some R-specific data structure.
- This can be achieved by parser functions such as `rvest::read_html()` or `XML::htmlparse`.

Parsing HTML with read_html

```r
library(rvest)
url <- "http://www.r-datacollection.com/materials/html/fort
myHTML<- read_html (url)
myHTML
```

```
## {html_document}
## <html>
## [1] <head>\n<meta http-equiv="Content-Type" content="text/htm
## [2] <body>\n<div id="R Inventor" lang="english" date="June/20
```

The Document Object Model

- The structure of the parsed HTML object can be better viewed using `xml_structure` function from the `xml2` package.

```
# Print the HTML excerpt with the xml_structure() function
xml2::xml_structure(myHTML)
```

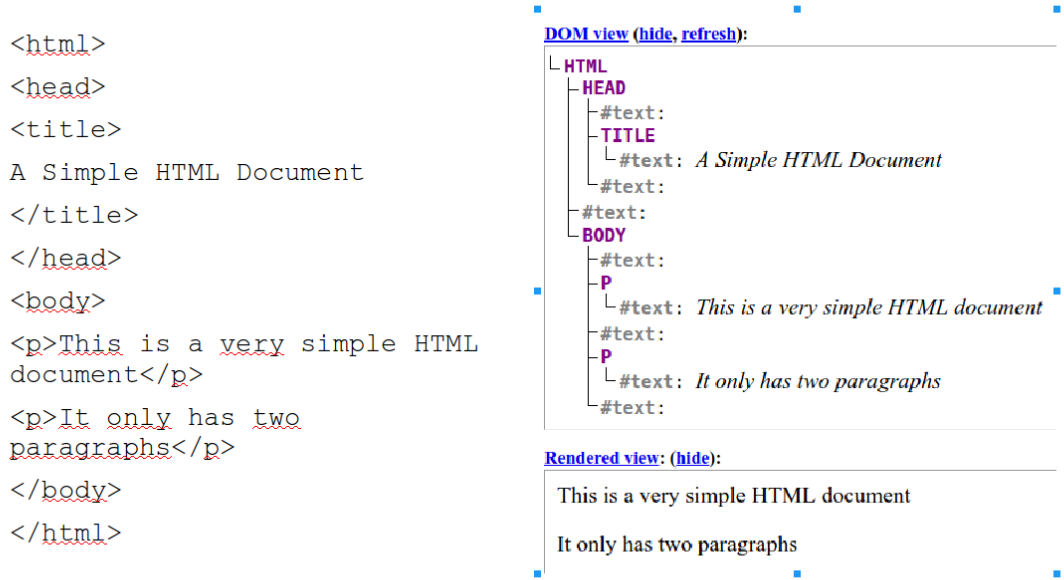
- This representation is related with what we call the *Document Object Model (DOM)*.

- A Document Object Model is a *queryable data object* that can be built from any HTML file and is useful for further processing of document parts.

A distraction: HTML tree structure

- A HTML document can be seen as a hierarchichal collection of tags which contain distinct elements.
- Hint: Paste the source code of the *fortunes.html* document in This viewer

```r
knitr::include_graphics("images/htmlHierarchy.png")
```



```html
<html>
<head>
<title>
A Simple HTML Document
</title>
</head>
<body>
<p>This is a very simple HTML
document</p>
<p>It only has two
paragraphs</p>
</body>
</html>
```

DOM view (hide, refresh):

```
└HTML
  ├HEAD
  │ ├#text:
  │ ├TITLE
  │ │ └#text: A Simple HTML Document
  │ └#text:
  ├#text:
  └BODY
    ├#text:
    ├P
    │ └#text: This is a very simple HTML document
    ├#text:
    ├P
    │ └#text: It only has two paragraphs
    └#text:
```

Rendered view: (hide):

This is a very simple HTML document

It only has two paragraphs

DOM-style parsers

- Transformation from HTML code to the DOM is the task of a *DOM-style parsers*.

- There are two mainstream packages that can be used for parsing HTML code
  - rvest package by Hadley Wickam,
  - XML package by Duncan Temple and Debbie Nolan.

- A few others can be found at CRAN Task View: Web Technologies and Services.

Scrapping tools (I): The `XML` package

- The `XML` package provides an interface to `libxml2` a powerful parsing library written in C.
- The package is designed for two main purposes
  - parsing xml / html content
  - writing xml / html content (*we wonn't cover this*)

What can be achieved with `XML`?

- The `XML` package is useful at 4 major types of tasks:

  1. parsing xml / html content
  2. obtaining descriptive information about parsed contents
  3. navigating the tree structure (ie *accessing its components*)
  4. querying and extracting data from parsed contents

- The `XML` package can be used for both XML and HTML parsing.

Parsing HTML with `rvest`

Scraping tools: The `rvest` package

- `rvest` is an R package written by Hadley Wickam to *easily scrap web pages*
  - Wrappers around the 'xml2' and 'httr' packages to make it easy to download, and manipulate, HTML and XML
  - It is inspired in the BeautifulSoup python package.
  - It is designed to work with magrittr to simplify tasks.
- See more information on `rvest` at:
  - rvest package on CRAN
  - rvest documentation on DataCamp

Basic `rvest` capabilities

- Get the data: Parse an html document from a url, a file on disk or a string containing html with `read_html()` (from the `xml2` package!). +info

- Extract elements using `html_element(s)()`. +info

- Use `html_text2()` to extract the plain text contents of an HTML element. +info

- Or use `html_attr(s)()` to retrieve the value of a single attribute. +info

- Use `html_table` to read a table from within a page. +info

More `rvest` capabilities

- Get children from an element `html_children()`.
- Extract, modify and submit forms with -`html_form()`, `set_values()` and `submit_form()`.
- Detect and repair encoding problems with:
  - `guess_encoding()` and `repair_encoding()`. Then pass the correct encoding into `html()` as an argument.

Examples (1): Read HTML

```
html_0 ← '
<html>
  <body>
    <h1>Web scraping is coo
    <p>It requires getting
    <p><a href="https://asp
  </body>
</html>'
```

- HTML data can be read with `read_html`.

```
html_object ← xml2::read_h
show(html_0)
```

XML structure can be better viewed with:

```
# Print the HTML excerpt wi
xml_structure(html_0)
```

Examples (2): html_elements()

```
list_of_links <- '<h3>Usefu
<ul>
  <li><a href="https://wiki
  <li><a href="https://www.
  <li><a href="https://diba
</ul>'
```

Extract all the "a" nodes from the bulleted list.

```
links <- list_of_links %>%
  read_html() %>%
  html_elements("a")
```

Examples (3): html_table()

```
sample1 <- minimal_html("<t
  <tr><th>Col A</th><th>Col
  <tr><td>1</td><td>x</td><
  <tr><td>4</td><td>y</td><
  <tr><td>10</td><td>z</td>
</table>")
```

```
sample1 %>%
  html_element("table") %>%
  html_table()
```

```
## # A tibble: 3 × 2
##   `Col A` `Col B`
##     <int> <chr>
## 1       1 x
## 2       4 y
## 3      10 z
```

Examples (3b): more `html_table()`

```r
url ← "https://en.wikipedia.org/wiki/List_of_World_Heritag
pageTables ← read_html (url)  %>%
  html_elements("table") %>%
  html_table()
M2← pageTables[[2]]
head(M2, n=3)
```

```
## # A tibble: 3 × 9
##   Name                            Image Location
##   <chr>                           <lgl> <chr>
## 1 Abu Mena                        NA    EgyAbusir,&nbsp;Egypt
## 2 Air and Ténéré Natural Reserves NA    Niger1Arlit Departmen
## 3 Ancient City of Aleppo          NA    Aleppo Governorate, &
## # … with abbreviated variable names ¹Criteria, ²`Areaha (acre
```

Using CSS selectors to locate information

Improving location using css selectors

- Functions such as `html_elements` or `html_table` return one or all the elements of a given kind.
- To decide *which objects to select* we must identify them.
- This may be done using CSS selectors that have been used in the page to give structure ("tags") or change properties ("class", "id") of objects.

Examples 4: Selection with tags

- We can select the elements of a given type letting `html_elements` know which type it is.

```
  myHTMLdoc <- '<html>
  <body>
    <div>Python </div>
    <p> Is perfect for programming.</p>
    <p> A nicely built language </p>
    <div>R </div>
    <p>Better for data analysis.</p>
    <p>Has prettier charts, too.</p>
  </body>
</html>'
```

```
theLanguages <- read_html(myHTMLdoc) %>%
  html_elements('div') %>%
  html_text2()
theLanguages
```

```
## [1] "Python" "R"
```

Examples 4b: Multiple selection

- The same idea can be used to select elements that have one of several tags

```
  myHTMLdoc <- '<html>
  <body>
    <div>Python </div>
    <p> Is perfect for programming.</p>
    <small> A nicely built language </small>
    <div>R </div>
    <p>Better for data analysis.</p>
    <small>Has prettier charts, too.</small>
  </body>
</html>'
```

```
theLanguages <- read_html(myHTMLdoc) %>%
  html_elements('div, small') %>%
  html_text2()
theLanguages
```

```
## [1] "Python"                    "A nicely built language"
## [4] "Has prettier charts, too."
```

Examples 5: Selection with class/id

- After inspecting the page it can be seen that the table we are interested in is of class "wikitable"
- This is informed to `html_element` as: *type.class*

```
url <- "https://en.wikipedia.org/wiki/List_of_World_Heritage_in_Danger"
oneTable <- read_html (url)  %>%
  html_element("table.wikitable") %>%
  html_table()
head(oneTable, n=3)
```

```
## # A tibble: 3 × 9
##   Name                            Image Location                 Crite…¹ Areah…² Year …³ Endan…⁴ Reason Refs
##   <chr>                           <lgl> <chr>                    <chr>   <chr>     <int> <chr>   <chr>  <chr>
## 1 Abu Mena                        NA    EgyAbusir,&nbsp;Egypt.mw-par… Cultur… 182 (4…    1979 2001–   "Cave… [17]…
## 2 Air and Ténéré Natural Reserves NA    Niger1Arlit Department,… Natura… 7,736,…    1991 1992–   "Mili… [20]…
## 3 Ancient City of Aleppo          NA    Aleppo Governorate, &nbsp;Sy… Cultur… 350 (8…    1986 2013–   "Syri… [22]
## # … with abbreviated variable names ¹Criteria, ²`Areaha (acre)`, ³`Year (WHS)`, ⁴Endangered
```

Combining selectors

- Selectors can be combined using operators as follows:

```
selector1 {space▷|+|~} selector2
```

- There are four types of combinators

    - `space`: Descendant combinator
    - `>`: Child combinator
    - `+`: Adjacent sibling combinator
    - `~`: General sibling combinator

Examples 6: Combining selectors

```
myhtml <- "<html>
<body>
<div class = 'first'>
<a>A link.</a>
<p>The first paragraph with
<a>another link</a>.
</p>
</div>
<div>
Not an actual paragraph,
but with a <a href='#'>link</a>.
</div>
</body>
</html>"
```

```
htmlObj <- myhtml %>% read_html()
htmlObj %>%
  html_elements('div.first a')
htmlObj %>%
  html_elements('div.first > a')
htmlObj %>%
  html_elements('div.first + div')
htmlObj %>%
  html_elements('div.first ~ div')
```

Examples 7: Combining selectors

```
myhtml<- '<html>
  <body>
    <div class="first section">
      Some text with a <a href="#">link</a>.
    </div>
    <div class="second section">
      Some text with <a href="#">another lin
      <div class="first paragraph">Some text
      <div class="second paragraph">Some mor
        <div>...</div>
      </div>
    </div>
  </body>
</html>'
```

- Select all divs that descend from another div.
- This can be done easily:

```
htmlObj<- myhtml %>% read_html()
# Select the three divs with a simple select
htmlObj %>%
    html_elements('div div')
```

- Or more complicated:

```
# ComplexSelect
htmlObj %>%
    html_elements('.first + .second > div,
                  div.second.paragraph > div
```

References and Resources

Resources