Piping and Webscrapping

James Balamuta

Department of Informatics, Statistics
University of Illinois at Urbana-Champaign

July 13, 2017

On the Agenda

- Piping Operator
  - Motivation
  - History
  - Details %>%
- Web Scrapping
  - Understanding Document Object Model
  - Using `rvest`

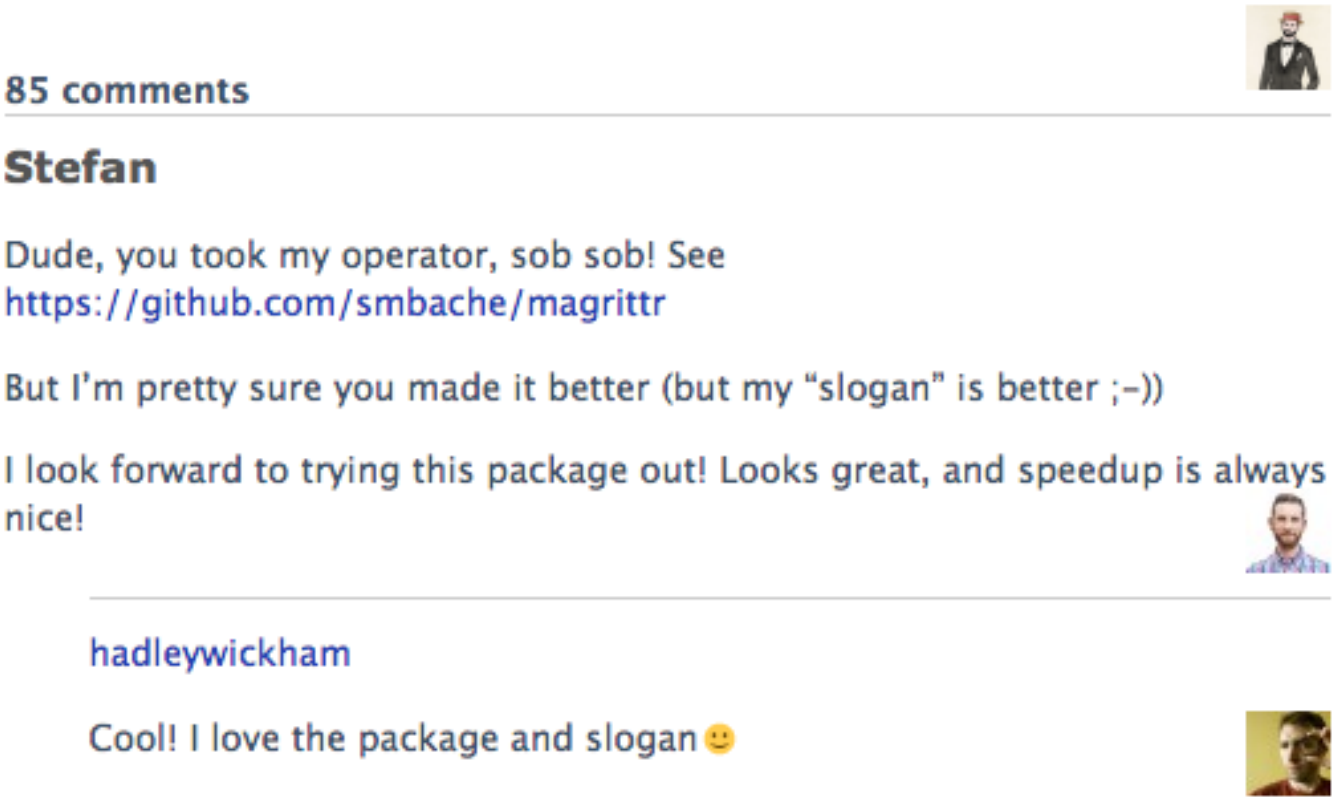
On the Agenda

History of the Piping Operator

The piping operator has existed in many forms over the years...

- Shell/Terminal: Pass command from one to the next with pipeline character |.
- `F#`: Forward pipe operator |> and served as the motivation for *R*'s.
- `Haskell`: Contains many piping operations derived from shell/terminal.
- `Python`: **Lacks** a similar implementation to *R*'s. The closest after 4 years appears to be in the toolz module.
- `R`: Stefan Milton Bache created %>% in the magrittr package.
  - Unbeknowist to Hadley, he introduced this functionality via %.% in his rewrite of `plyr` called `dplyr` to which Stefan famously replied...

Origins of the Pipe Operator in R



85 comments

Stefan

Dude, you took my operator, sob sob! See https://github.com/smbache/magrittr

But I'm pretty sure you made it better (but my "slogan" is better ;-))

I look forward to trying this package out! Looks great, and speedup is always nice!

hadleywickham

Cool! I love the package and slogan

Stefan Milton Bache commenting on Hadley's Introducing `dplyr` post on the RStudio Blog.

Remember

In English: **This is not a pipe**. Follows from René Magritte's The Treachery of Images

Present Day...

Up till now, if we wanted to have step-wise operations on reading data in we would need to do:

```
input_data = read.csv('/path/to/data.csv')
subset_data = subset(input_data, treatment > 10)
top_20_data = head(subset_data, 20)
```

Under this approach, we have successfully littered the global environment with tons of variables that have a one time only use.

Long and painful...

To get around that, we can embed the function calls.

```
top_20_data = head(
                    subset(
                           read.csv('/path/to/data.csv'),
                           treatment > 10),
                  20)
```

Though, that doesn't look very nice and the logic is *hard* to follow...

Enter the Pipe Operator

To simplify the process, we opt ot use a *pipe* operator defined as %>% in the `magrittr` package.

Examples:

- `x %>% rfunction`
  - Same as `rfunction(x)`
- `x %>% rfunction(arg = value)`
  - Same as `rfunction(x, arg = value)`

Piping is Sequential Logic

Take for example ordering a Starbucks drink via Mobile Order:

find drink, select store, order, go to store, pick up coffee.

```
pickup(goto(store(drink("Java Chip Frap"),
                  loc="Green St.")))
```

Or

```
"Java Chip Frap" %>% drink %>%
  store(loc="Green St.") %>%
  goto %>%
  pickup
```

Switching to the Pipe

Old:

```
top_20_data = head(
                    subset(
                           read.csv('/path/to/data.csv'),
                           treatment > 10),
                  20)
```

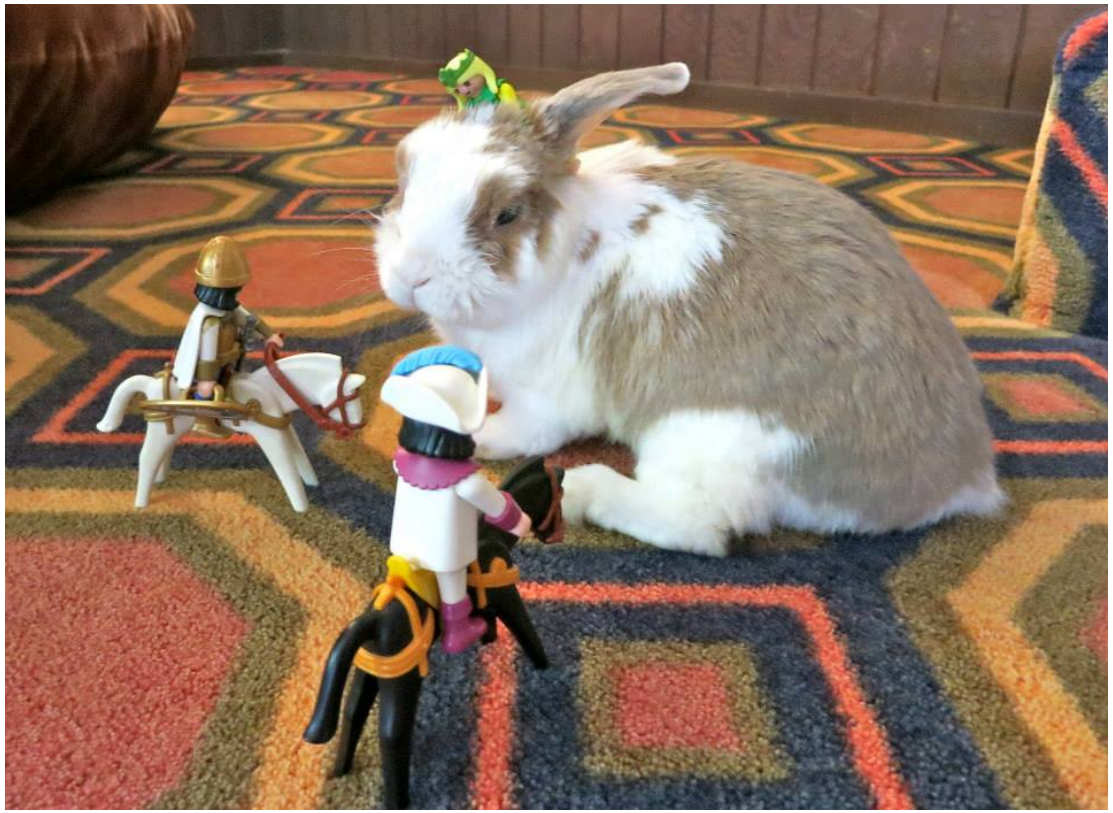
New:

```
read.csv('/path/to/data.csv') %>%
  subset(treatment > 10) %>%
  head(20) -> top_20_data
```

Is the Piping Operator a save all?

- **No.**
- *However*, the pipe is probably the *most* significant operator to move into *R*'s ecosystem since 2014 since it makes *R* code more user friendly.
- The operator is **not** for internal package development as it makes for harder debugging.

Bunny Foo Foo and Piping



Hadley Wickham's Bunny Foo Foo Example during his keynote at UseR 2016!

- Clip starts at 33m 48s and goes till 36m 30s...
- Did you read about this example within the Pipes chapter in R for

Example Piping Data

For the next few examples, we'll use the following simulated data:

```
# Set Seed for Reproducibility
set.seed(1123)

# Generate Data
d = data.frame(x=rnorm(10), y = rnorm(10))
```

Problems associated with Piping: Argument Order

`x` may *not* be the first function parameter. e.g.

```
myfunc = function(other_param, x)
```

To get around this issue, use the `.` character to redirect pipe input to a different argument. So, in `magrittr` land, the period or `.` is how to indicate a redirection.

Problems associated with Piping: Argument Order

For example, when modeling with `lm` notice:

```
# Moved `d` to the data argument
d %>% lm(y ~ x, data = .)
```

```
##
## Call:
## lm(formula = y ~ x, data = .)
##
## Coefficients:
## (Intercept)            x
##     -0.1744       0.1695
```

Problems associated with Piping: Extracting Information

Sometimes, you may wish to only be able to extract the *n*th element and pass that further along in the chain.

Use the . operator to represent the object on the right hand side (RHS) of the pipe within the left hand side (LHS).

```
d %>% .[["y"]]
```

```
##   [1]  0.31234925 -0.17384622  0.05596198  0.79823340
##   [5] -0.69309540 -0.17145346 -1.15644219  1.91378654
##   [9] -1.21734666 -0.80973361
```

Problems associated with Piping: The Tee Operator

Sometimes a function might not return a value and you want the chain to continue on the previous computation.

Examples:

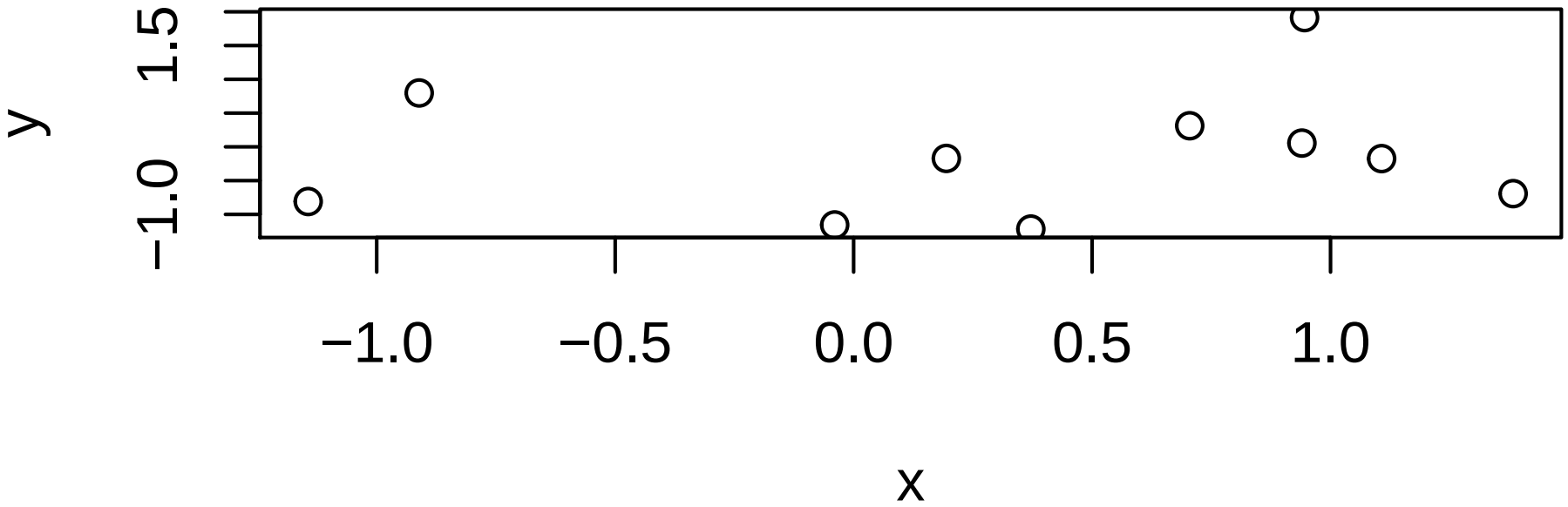
- Printing
- Plotting
- Saving

In such cases, the "tee" operator given as %T>% should be used.

%T>% returns LHS value instead of the RHS operation result to the next part of the chain. So, it "skips" sending output from one portion of the chain to the next.

Problems associated with Piping: The Tee Operator

```
d %>%
as.matrix %T>%
plot %>% # plot will not return anything
colSums  # as.matrix goes into colSums.
```




```
##          x          y
##   3.552043 -1.141586
```

Exercises

1. Make the following "pipeable"

```
tail(subset(iris, Petal.Width > mean(Petal.Width)))
```

2. Write a pipe that provides the `sqrt` of 2+2
3. Create another pipe that transforms two strings into one **upper** case string.

```
a = "stat 385 is evolving"
b = "My pokemon is evolving faster..."
```

Summary

- Piping is a powerful tool
- Try to design functions so that they are "interconnected"
- Avoid using the piping operator within a package's internals.

On the Agenda

1. Piping
   - History
   - Background
   - magrittr
   - Examples
   - Change argument order
   - Extract Value
   - Carrying Over Data
2. Web Scrapping
   - Background
   - HTML Overview
   - Scrappers in R
   - rvest

Web Scraping

Definition:

Web scraping (**web harvesting** or **web data extraction**) is a computer software technique of extracting information from websites.

From https://en.wikipedia.org/wiki/Web_scraping

A primer on HTML

Before we begin, we need to talk about the language of the web:

HTML

- **HTML** stands for **H**yper **T**ext **M**arkup **L**anguage.
- This is remarkably different from **Markdown**, which wants the minimalist amount of content declaration

A primer on HTML

The basic structure or *markup* is:

```
<!DOCTYPE html>
<html>
<head>
<title>Title of Page</title>
</head>
<body>

<h1 align = "center">First order heading (large)</h1>
<p>Paragraph for text with a
   <a href="http://www.stat.illinois.edu">link!</a>
</p>

<!-- Comment -->

</body>
</html>
```

HTML Tags

An **HTML** tag is given as:

```
<tag>content</tag>
```

In the case of **bold** text it would be:

```
<b>some text that I want bold</b>
```

Question: What would happen if we did not close the tag?

HTML Attributes

Attributes allow for additional information to be embedded along side content.

```
<tag attribute="property">content</tag>
```

For example, the hyperlink or more precisely a link is defined as:

```
<a href="http://illinois.edu">UIUC Website</a>
```

where

- `href="link"` indicates the URL location the link points to.

HTML Reference

Outside of that brief introduction, you should definitely consider learning more about **HTML** via:

- W3Schools
- Mozilla's HTML Reference Guide

Web Scrapping Packages in R

There are many packages in *R* that provide web scraping functionality:

- `rvest` by Hadley Wickham
  - Downloads HTML and parses it. Support exists for user sessions
- `RSelenium` by John Harrison
  - Opens an installed web browser and controls the interface.
  - Use this when `rvest` fails completely due to websites requiring `JavaScript`.
- `xml2` by Hadley Wickham and Gang
  - Primarily an xml reader that can now also write xml (in turn also HTML).
- `XML` by Duncan Temple Lang
  - Original XML reader that has survived the test of time.

Focusing...

For simplicity, we will focus our attention on `rvest` by Hadley.

Software

- Chrome Web browser
- SelectorGadet
  - Drag the URL to the bookmark bar for a "SelectionGadget"
  - Or download the Chrome extension
- For help, please see the SelectorGadget vignette
  - For a local copy in *R*, type: `vignette("selectorgadget")`
- Alternatively, we can use Chrome's Built in Developer Tools via:
  - Windows: `Ctrl` + `Shift` + `C`
  - macOS: `Command` + `Shift` + `C`

Core rvest functions

Within here are the key webscrapping functions you will likely use.

Function	Description
`read_html()`	Download HTML Output from a website and read into R
`html_nodes`	Extract HTML Nodes given by `<tag></tag>`
`html_table`	Convert an HTML table (`<table></table>`) into a `data.frame` object
`html_text`	Extract the text between an HTML tag `<tag>content</tag>`

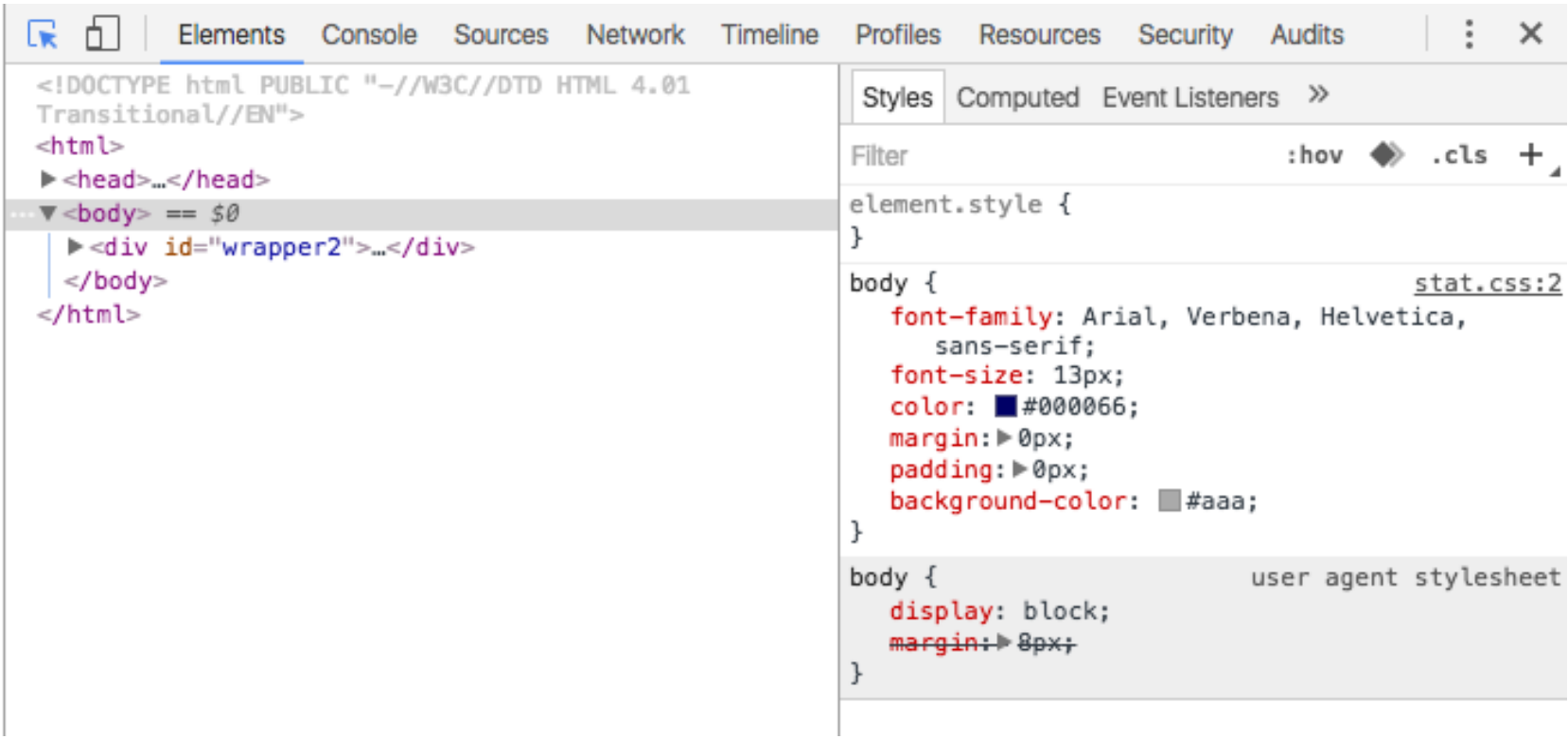
Simple `rvest` example - Directory of PhD student information

Let's focus a bit on scrapping the `https://www.informatics.illinois.edu/` website

Specifically, we're going to visit the PhD Student Directory

Simple `rvest` example - Finding the Selectors

- Go to the PhD Student Directory
- Open Chrome's Dev Tool
  - Windows: `Ctrl` + `Shift` + `C`
  - macOS: `Command` + `Shift` + `C`

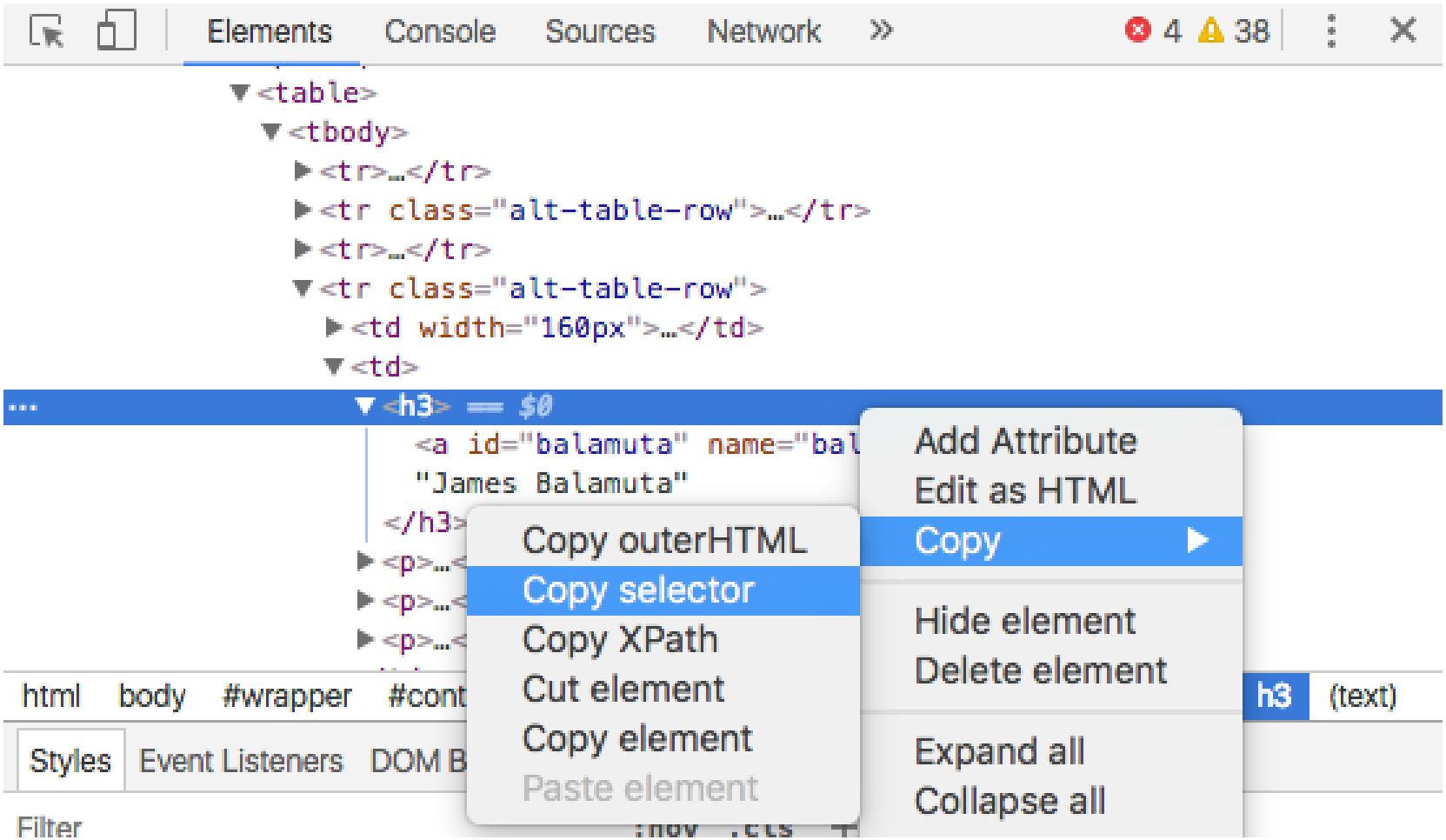


Simple `rvest` example - Finding the Selectors

- Click on "James Balamuta" under the shortcut

Simple `rvest` example - Finding the Selectors

- Right click on the element to bring up a `Copy` menu and select `Copy Selector`



Simple `rvest` example - Finding the Selectors

This gives:

```
#main > article > section > table:nth-child(4) > tbody > tr:nth-child(4) > td:nth-child(2) > h3
```

To generalize it, we'll aim to drop the `nth-child(4)` selector on `tr`

```
#main > article > section > table:nth-child(4) > tbody > tr > td:nth-child(2) > h3
```

Questions:

1. What do you think the `nth-child` operator does?
2. Why is a # next to `main` but not `table`?

Simple rvest example

Scrapping the Directory of PhD student information

```
# Load the Package
library("rvest")
```

```
# Grab a copy of the PhD Directory
phds = read_html(
  "https://www.informatics.illinois.edu/people-2/phd-students/")

# Retrieve PhD Table Entries
phds %>%
  # Uses selector given before
  html_nodes("#main > article > section > table:nth-child(4) >
              tbody > tr > td:nth-child(2) > h3") %>%
  html_text() -> phd_names

phd_names %>% .[[4]]
```

```
## [1] "James Balamuta"
```

More complex operations

Sometimes you will need to extract information directly within the tag.

Here is a set of "ideal" functions for that.

Function	Description
`html_name`	Obtain the name of the tag e.g. `<h1></h1>` gives `h1`
`html_attrs`	Obtains all the attributes of the tag
`html_attr`	Obtain only the value associated with a specific attribute.

Obtaining NetIDs

```
# Get a list of PhD Names
phds %>%
  # Modify selector
  html_nodes("#main > article > section > table:nth-child(4) >
              tbody > tr > td:nth-child(2) > p > a") %>%
  # Get the linking information
  html_attr("href") %>%
  # Find only entries with mailto:
  grep("mailto:", x = ., value = T) %>%
  # Remove everything prior to the directory call
  gsub("mailto:(.*)@.*","\\1", x = .) -> phd_netids

phd_netids %>% .[[4]] # Pop the fourth ID
```

```
## [1] "balamut2"
```

User Sessions

Often, you might need to create a *persistent* instance where you can make requests to a webserver and receive information. Here, you will find an overview of managing such a session.

Function	Description
`html_session`	Creates an HTML Session that has persistent cookies.
`jump_to`	Switches the session from being on one page to the next
`follow_link`	Enables the session to follow a specific link on a given page
`back`	Navigates the session back to the prior page
`forward`	Moves the browser forward to the next page

User Sessions - Example

```
# Similar to the `read_html`
coatless = html_session("http://github.com/coatless")

# Notice I'm resaving into `coatless`
coatless %>%
  follow_link("thecoatlessprofessor") -> coatless
```

```
## Navigating to http://thecoatlessprofessor.com
```

```
# Go back to GitHub (not saved)
coatless %>% back()
```

```
## <session> https://github.com/coatless
##   Status: 200
##   Type:   text/html; charset=utf-8
##   Size:   95583
```

User Session - continued

```
# Go to one of the repositories
coatless  %>% back()  %>% follow_link("@tmsalab")
```

```
## Navigating to https://github.com/tmsalab
```

```
## <session> https://github.com/tmsalab
##   Status: 200
##   Type:   text/html; charset=utf-8
##   Size:   38978
```

Exercises

1. Obtain the first news story title from `https://news.google.com/`
2. Find the top listed stars of The Thomas Crown Affair
3. Obtain the Statistics faculty netid from `http://www.stat.illinois.edu/people/faculty.shtml`

Summary

- `rvest` is small but powerful.
- Be mindful of the HTML tags.