Webscraping and the Text Analysis Pipeline

Computational Text Analysis

Theresa Gessler

University of Zurich | http://theresagessler.eu/ | @th_ges

2022-05-12

Program

scraping: /ˈskreɪpɪŋ/, *to remove (an outer layer, for example) from a surface by forceful strokes of an edged or rough instrument*

web scraping: to collect data from the web by removing the unnecessary parts (sometimes with a rough instrument)

→ Project to use this for text analysis

Program

- downloading data from simple HTML pages
  - texts
  - tables
  - iteration
- other scraping scenarios
  - APIs
  - dynamic pages
  - webcrawlers / spiders
  - task scheduling
  - (importing offline data)

Scraping Data from simple pages

Simple pages

Scraping

- extracting data from webpages
  - anything from university webpage to social media
  - lots of different techniques
- today: **scraping simple static pages**
- types of scraping
  - structured vs. **unstructured data**
    - *scenario 2: APIs*
  - **accessible** vs. protected data
    - *scenario 3: dynamic pages*
  - gathering as diverse information as possible from different pages vs. **very specific scrapers**
    - *scenario 4: web crawling*
  - **one-off scraping** vs. regular data collection
    - *scenario 5: task scheduling*

Simple pages

Scenario 1: Static pages (what we'll cover)

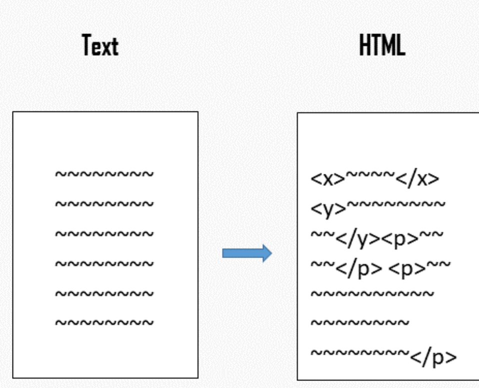
- extracting text, links and tables from many standard webpages
- conditions
  - page written in HTML / XML
  - page has a static URL through which you can reach it

Procedure

- 'parsing' page in R
- extracting relevant parts
- cleaning into usable format (e.g. data frame, raw text, …)

Simple pages

Simple pages



Simple pages

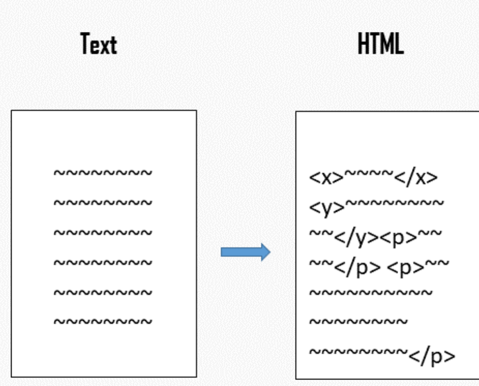
HTML

- **H**yper **T**ext **M**arkup **L**anguage
  - *markup*: additional description of formatting beyond the content of the text
- language consists of **HTML tags** to specify character / behaviour of text
- HTML tags typically consist of a starting and an end tag (exceptions: images, line breaks etc.)
  - many exceptions where it is not 'markup'
- they surround the text they are formatting

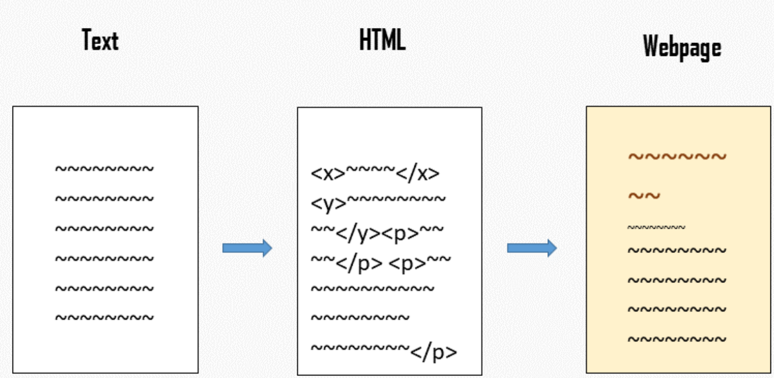
Example:

```
<tagname>Content goes here...</tagname>
```

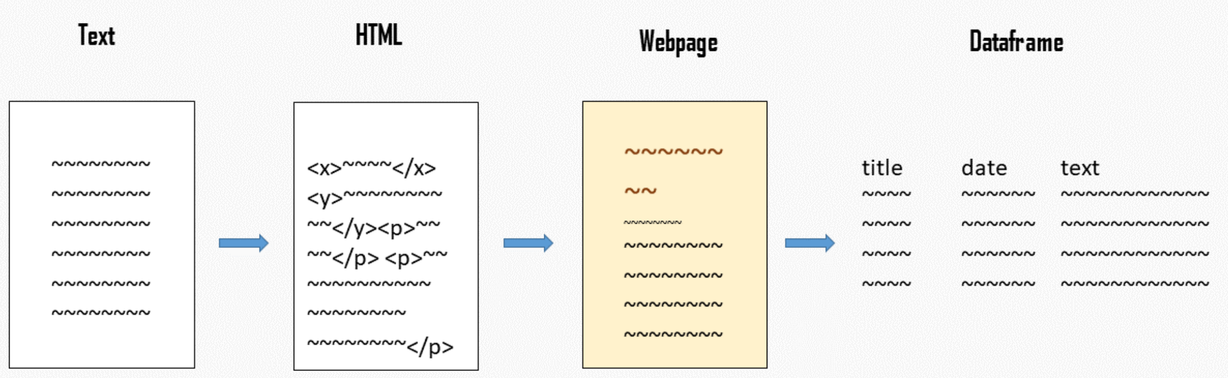
Simple pages



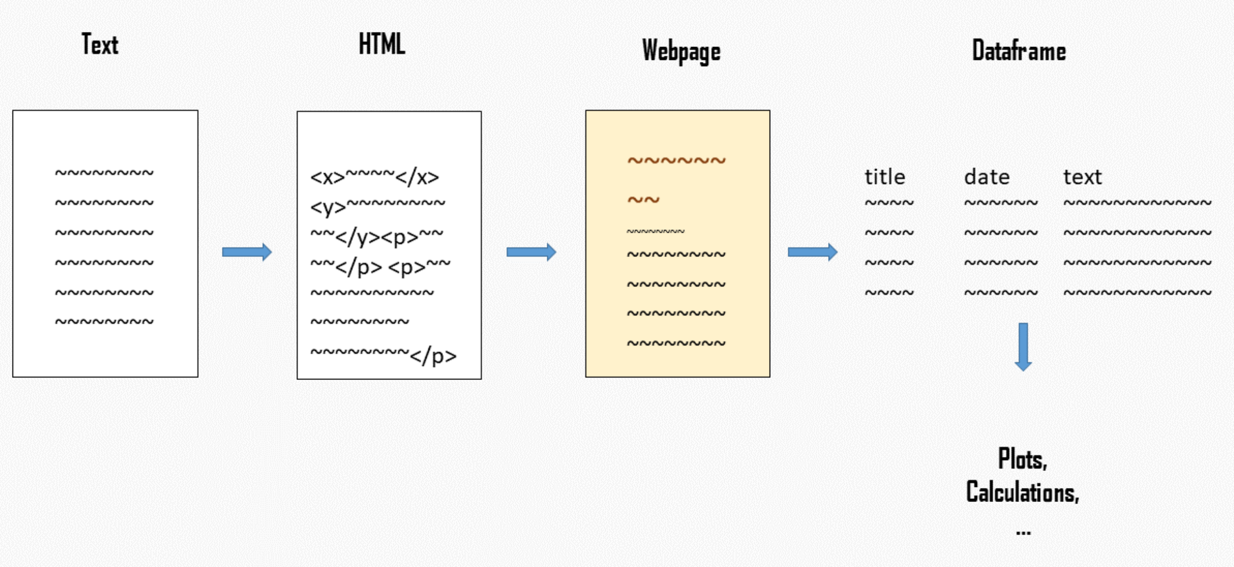
Simple pages



Simple pages



Simple pages



Simple pages

Webscraping HTML Pages

- collecting data from HTML pages means removing the formatting but keeping any information it contains
  - 'parsing' of page structure
  - 'selecting' of parts of pages

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Quotes to Scrape</title>
    <link rel="stylesheet" href="/static/bootstrap.min.css">
    <link rel="stylesheet" href="/static/main.css">
</head>
<body>
    <div class="container">
        <div class="row header-box">
            <div class="col-md-8">
                <h1>
                    <a href="/" style="text-decoration: none">Quotes to Scrape</a>
                </h1>
            </div>
            <div class="col-md-4">
                <p>

                    <a href="/login">Login</a>

                </p>
            </div>
        </div>


<div class="row">
    <div class="col-md-8">

    <div class="quote" itemscope itemtype="http://schema.org/CreativeWork">
        <span class="text" itemprop="text">“The world as we have created it is a process of our thinking. It cannot be changed without
changing our thinking.”</span>
        <span>by <small class="author" itemprop="author">Albert Einstein</small>
        <a href="/author/Albert-Einstein">(about)</a>
        </span>
        <div class="tags">
            Tags:
            <meta class="keywords" itemprop="keywords" content="change,deep-thoughts,thinking,world" /    >

            <a class="tag" href="/tag/change/page/1/">change</a>

            <a class="tag" href="/tag/deep-thoughts/page/1/">deep-thoughts</a>

            <a class="tag" href="/tag/thinking/page/1/">thinking</a>

            <a class="tag" href="/tag/world/page/1/">world</a>

        </div>
    </div>
```

Example HTML Code

Basic HTML tags

```
<html>
    <head>
        <title>Title of your web page</title>
    </head>
    <body>
    HTML web page content
    </body>
</html>
```

- we are mostly interested in what is inside the **body**, that is, the content of a webpage
- **head** gives meta information, often used by search engines
- tags can be **nested**

Basic HTML Tags: Headings

Headings are defined by numbered h tags. Examples (with code and outcome):

```
<h1> your heading</h1>
```

```
<h2> a smaller heading</h2>
```

a smaller heading

```
<h3> an even smaller heading</h3>
```

an even smaller heading

```
<h4> an even smaller heading</h4>
```

an even smaller heading

```
<h5> an even smaller heading</h5>
```

an even smaller heading

Basic HTML Tags: Paragraphs

Paragraphs are defined by `div` or `p` tags.

Examples:

```
<p>this is a paragraph.</p><p>and this is the next.</p>
```

this is a paragraph.

and this is the next.

```
<div>this is a paragraph.</div><div>and this is the next.</div>
```

this is a paragraph.
and this is the next.

Basic HTML Tags: Attributes

- All HTML elements can have attributes
- Attributes provide additional information about an element
  - they are included inside the tag

Usage

- they are always specified in the starting tag
  - e.g. `<title attribute="x"> Title </title>`
- Attributes usually come in name and value pairs
  - e.g. attributename="attributevalue"

Basic HTML Tags: Attributes - Links

- Most common case of attributes: **links**
  - text or images turned into a link by surrounding `<a>` tag (*anchor*)
  - link address specified as href attribute (*hyperreference*)

Example:

```
This is text <a href="http://quotes.toscrape.com/">with a link</a>.
```

This is text with a link.

Basic HTML Tags: Attributes

- other examples of attributes
    - src: location of an image
    - styles: formatting
    - classes: formatting for groups

Examples:

```
<img src="myimage.jpg">
```

```
<p style="color:red">This is a paragraph.</p>
```

This is a paragraph.

```
<p class="error">Red highlight</p>
```

Styling with Classes

Webpages like blogs often define **Styles** and apply them to classes across the whole webpage. This use of classes is very common because it reduces the risk of accidentally formatting one instance of a repeated element differently.

```
<style>
p.error {
  color: red;   border: 1px solid red;
}
</style>

<p class="error">Red highlight</p>
```

Red highlight

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Quotes to Scrape</title>
    <link rel="stylesheet" href="/static/bootstrap.min.css">
    <link rel="stylesheet" href="/static/main.css">
</head>
<body>
    <div class="container">
        <div class="row header-box">
            <div class="col-md-8">
                <h1>
                    <a href="/" style="text-decoration: none">Quotes to Scrape</a>
                </h1>
            </div>
            <div class="col-md-4">
                <p>

                    <a href="/login">Login</a>

                </p>
            </div>
        </div>


<div class="row">
    <div class="col-md-8">

    <div class="quote" itemscope itemtype="http://schema.org/CreativeWork">
        <span class="text" itemprop="text">“The world as we have created it is a process of our thinking. It cannot be changed without
changing our thinking.”</span>
        <span>by <small class="author" itemprop="author">Albert Einstein</small>
        <a href="/author/Albert-Einstein">(about)</a>
        </span>
        <div class="tags">
            Tags:
            <meta class="keywords" itemprop="keywords" content="change,deep-thoughts,thinking,world" /    >

            <a class="tag" href="/tag/change/page/1/">change</a>

            <a class="tag" href="/tag/deep-thoughts/page/1/">deep-thoughts</a>

            <a class="tag" href="/tag/thinking/page/1/">thinking</a>

            <a class="tag" href="/tag/world/page/1/">world</a>

        </div>
    </div>
```

Have another look at the webpage - do you understand more now?

rvest: The Swiss army knife of scraping

- r package for scraping
- **strengths**
  - covers most frequent use cases
  - integration with other packages, e.g. tidyverse
- **weaknesses**
  - relatively simple: no dynamic webpages

Main uses

- Tables
- Texts
- extracting links

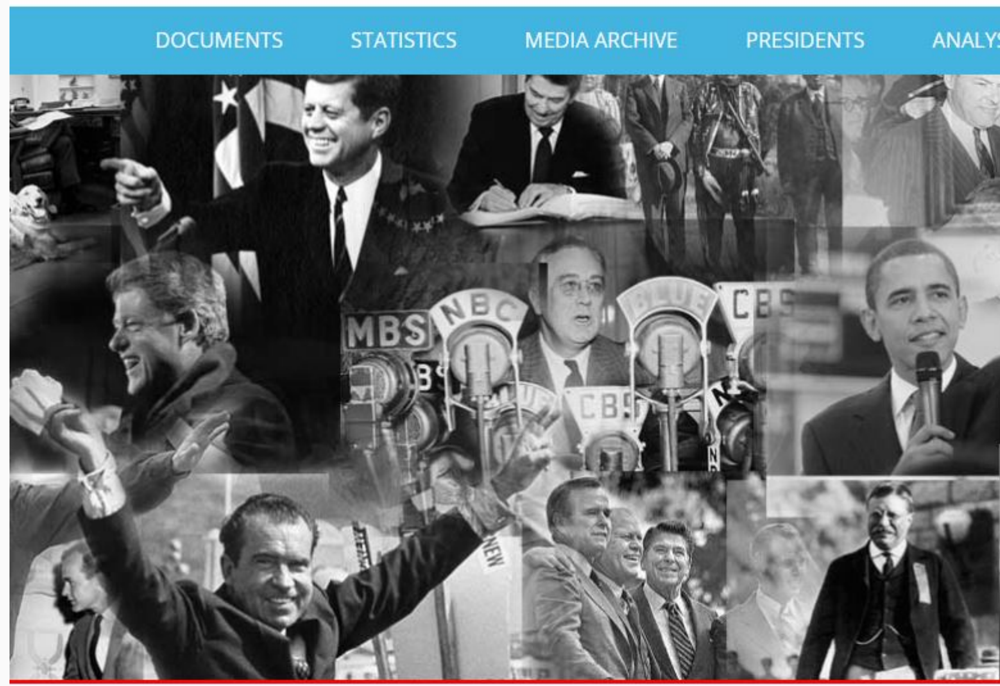
Overview of rvest commands

Limited set of commands:

- `read_html()`
- `html_elements()` (first occurence: `html_element()`)
- `html_text()`
- `html_table()`
- `html_attrs()` (first occurence: `html_attr()`)

The American Presidency Project

https://www.presidency.ucsb.edu/



'Parsing' HTML

- Example: https://www.presidency.ucsb.edu/
- scraping with R
    - you manually specify a resource
    - R sends request to server that hosts website
    - server returns resource
    - R parses HTML (i.e., interprets the structure), but does not render it in a nice fashion
    - you tell R which parts of the structure to focus on and what to extract

```
library(rvest)
url <- "https://www.presidency.ucsb.edu/"
page <- read_html(url) # returns parsed page
```

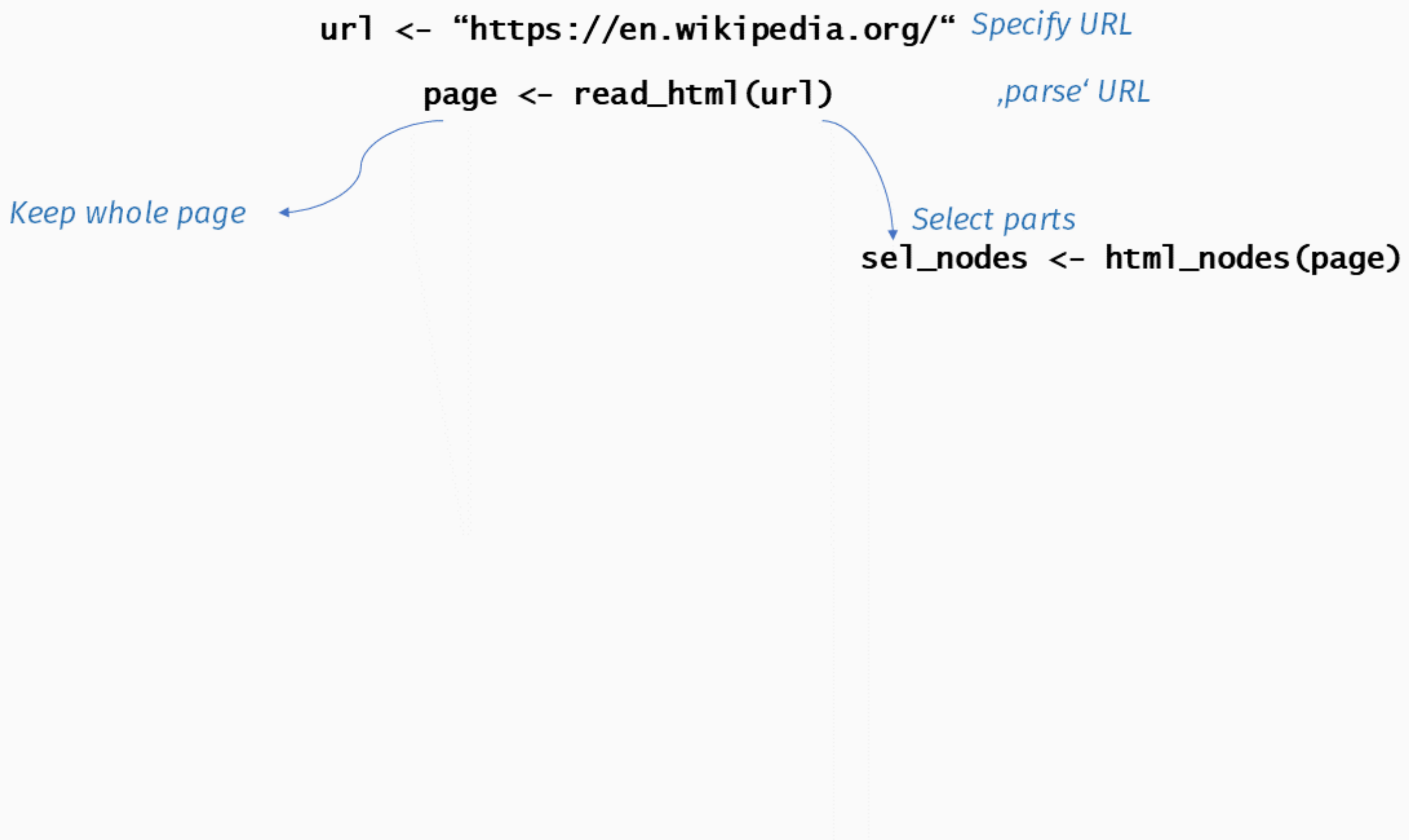
We practice this together in R

The process of scraping

```
url <- “https://en.wikipedia.org/“
```
Specify URL

```
page <- read_html(url)
```
‚parse‘ URL

The process of scraping



CSS Selectors

- we use the *appearance* / *style* of text to select specific parts
- based on specific **HTML elements**
  - tags
  - classes
  - attributes
- CSS selectors provide a *language* in which we can describe what we select at a more abstract level

text to be selected

"text to be selected" ← vs. → **text in red, surrounded by a red border**

Basic selectors

*	universal selector	Matches everything.	*
element	element / type selector	Matches an element	p
[attribute]	attribute selector	Matches elements containing a given attribute	[href]
[attribute=value]	attribute selector	Matches elements containing a given attribute with a given value	[href=/]
.class	class selector	Matches the value of a class attribute	.header
#id	ID selector	Matches the value of an id attribute	#first

More complex attribute selectors

[attribute*=value]	Matches elements with an attribute that contains a given value	a[href*="pressrelease"]
[attribute^="value"]	Matches elements with an attribute that starts with a given value	a[href^="/press/"]
[attribute$="value"]	Matches elements with an attribute that ends with a given value	[href$=".pdf"]

Combining CSS Selectors

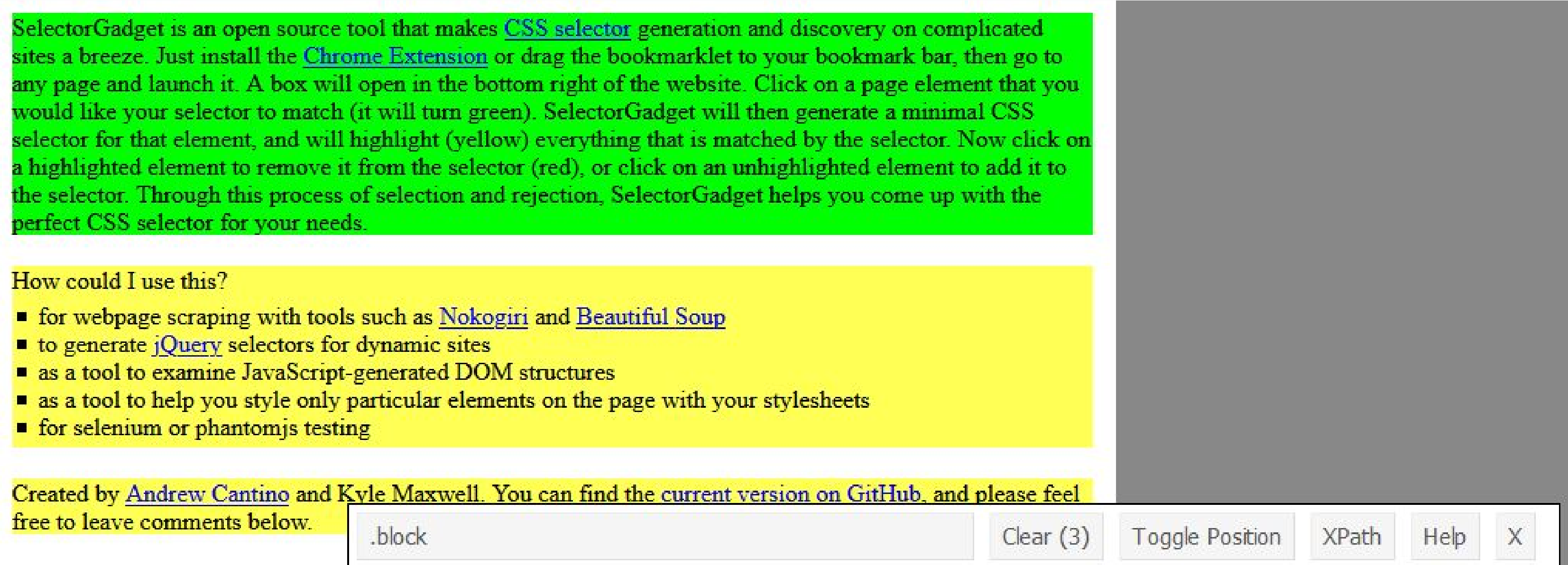
There are several ways to combine CSS Selectors:

element,element	Selects all <div> elements and all <p> elements	div, p
element element	Selects all <p> elements inside <div> elements	div p
element>element	Selects all <p> elements where the parent is a <>div> element	div > p
element+element	Selects all <p> elements that are placed immediately after <div> elements	div + p
element1~element2	Selects every element that are preceded by a <p> element	p ~ ul

Dine at the CSS Diner. And use SelectorGadget

Selectorgadget

- Selectorgadget
- Vignette



Extracting links from webpages

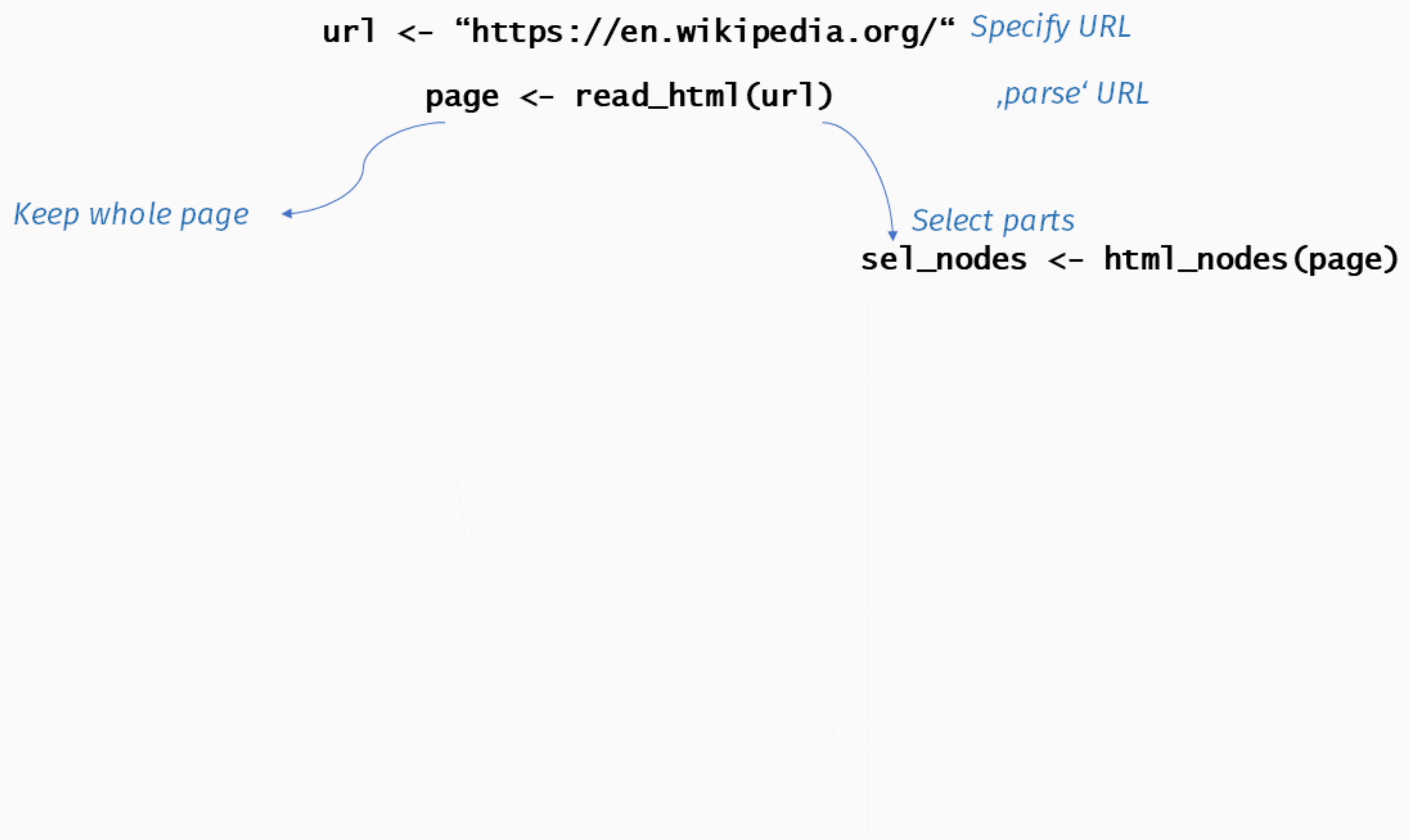
Extracting links

Extracting links from webpages

- unlike a book that we read cover to cover, webpages distribute information over multiple pages
- *hyperlinks* connect one page to the others → we follow them by clicking
    - we need to deal with this differently when scraping

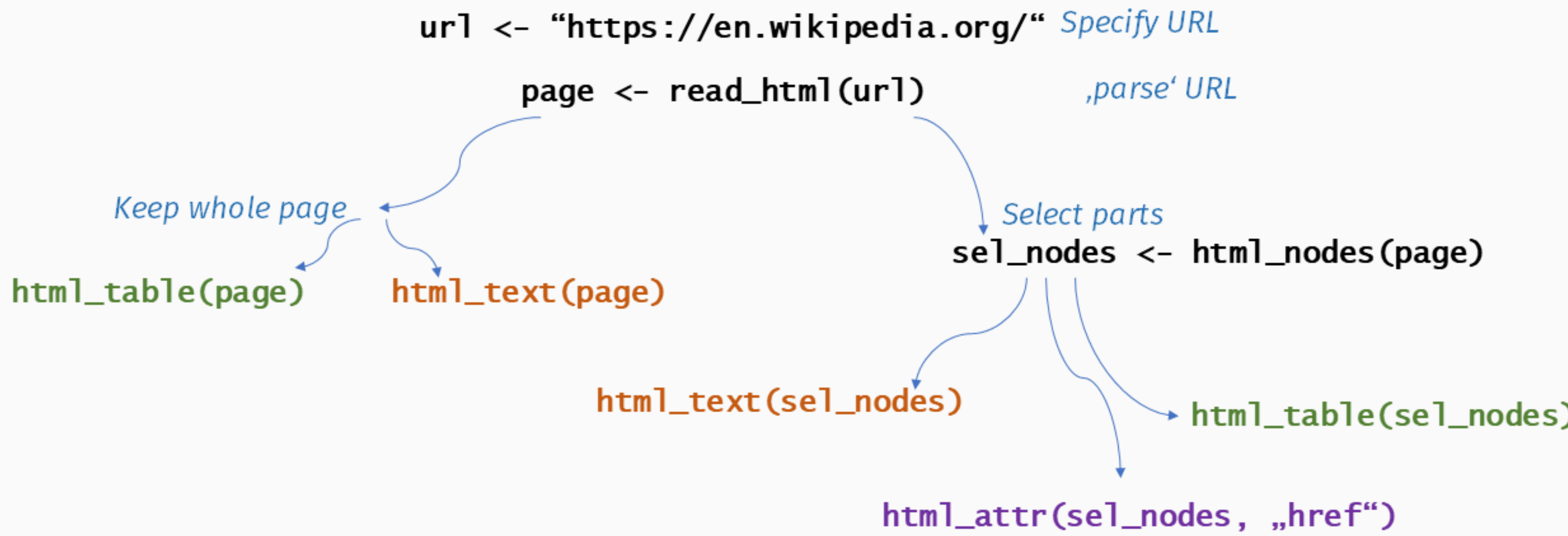
Extracting links

The process of scraping



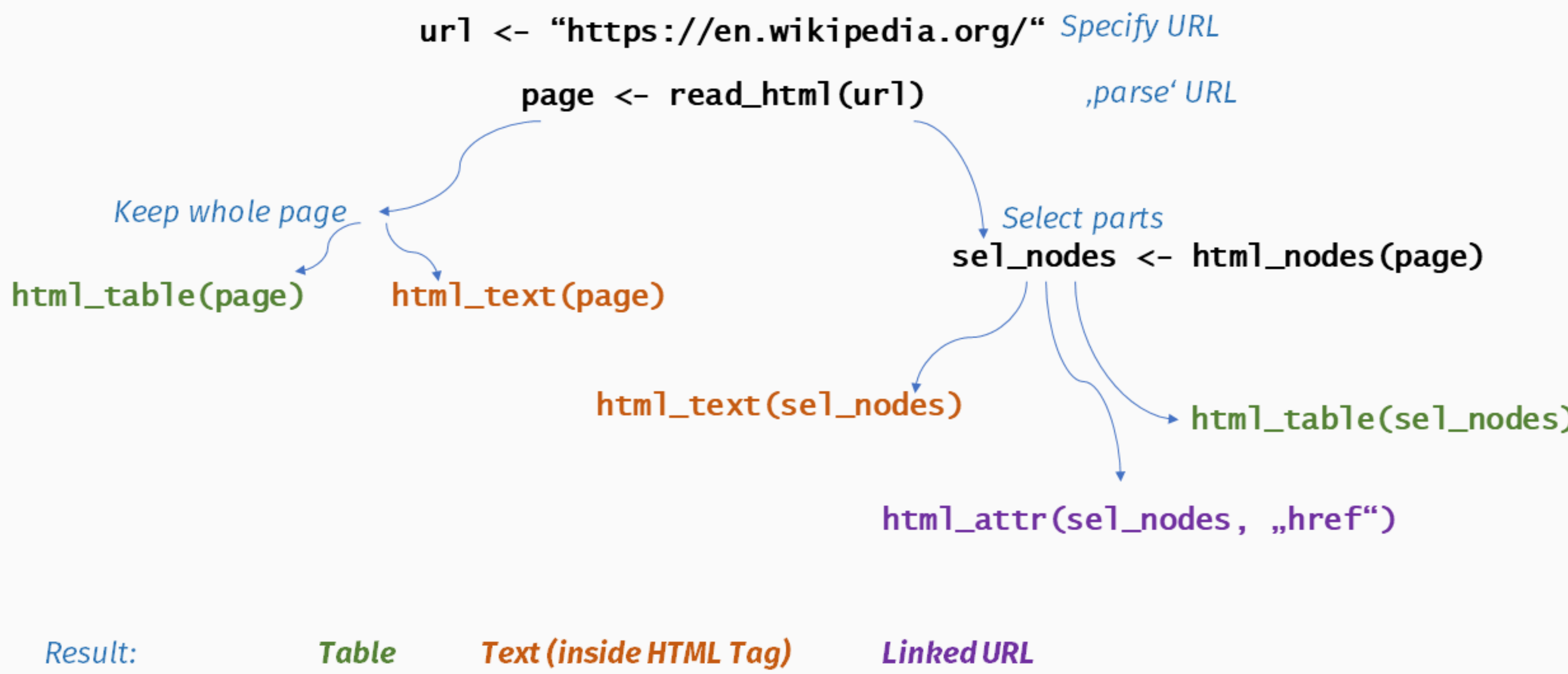
Extracting links

The process of scraping



Extracting links

The process of scraping



Extracting links

Links in HTML

- We discussed links as a common case of **attributes**
  - text (or other content) turned into a link with `<a>` tag (*anchor*)
  - link address specified as href attribute (*hyperreference*)

```
This is text <a href="https://www.presidency.ucsb.edu/">with a link</a>.
```

This is text with a link.

Extracting links

Extracting links with rvest

- extracting the text of the link
  - `html_elements(page,"a") %>% html_text()`
- extracting the attribute of the link (the hyperreference)
  - `html_elements(page,"a") %>% html_attr("href")`

```
page <- read_html('This is text <a href="https://www.presidency.ucsb.edu/">with a
link</a>.')
html_elements(page,"a") %>% html_text()
```

```
## [1] "with a link"
```

```
html_elements(page,"a") %>% html_attr("href")
```

```
## [1] "https://www.presidency.ucsb.edu/"
```

Caution: Link is attribute of `<a>`-Tag! `html_attr(page,"href")` **vs.** `html_elements(page,"a") %>% html_attr("href")`

Automation

Automation

Budapest (/ˈbuːdəpɛst/, Hungarian pronunciation: [ˈbudɒpɛʃt]) is the capital and the most populous city of Hungary, and the tenth-largest city in the European Union by population within city limits.[9][10][11] The city had an estimated population of 1,752,286 in 2019 distributed over a land area of about 525 square kilometres (203 square miles).[12] Budapest is both a city and county, and forms the centre of the Budapest metropolitan area, which has an area of 7,626 square kilometres (2,944 square miles) and a population of 3,303,786, comprising 33 percent of the population of Hungary.[13][14]

https://en.wikipedia.org/wiki/Help:IPA/English
https://en.wikipedia.org/wiki/Help:IPA/Hungarian
https://en.wikipedia.org/wiki/Capital_city
https://en.wikipedia.org/wiki/List_of_cities_and_towns_of_Hungary
https://en.wikipedia.org/wiki/Hungary
https://en.wikipedia.org/wiki/Largest_cities_of_the_European_Union_by_population_within_city_limits
https://en.wikipedia.org/wiki/European_Union
https://en.wikipedia.org/wiki/Budapest#cite_note-TIME2-9
https://en.wikipedia.org/wiki/Budapest#cite_note-10
https://en.wikipedia.org/wiki/Budapest#cite_note-11
https://en.wikipedia.org/wiki/Budapest#cite_note-Encarta-12
https://en.wikipedia.org/wiki/List_of_cities_and_towns_of_Hungary
https://en.wikipedia.org/wiki/Counties_of_Hungary
https://en.wikipedia.org/wiki/Budapest_metropolitan_area
https://en.wikipedia.org/wiki/Budapest#cite_note-13
https://en.wikipedia.org/wiki/Budapest#cite_note-14

Automation

Budapest (/ˈbuːdəpɛst/, Hungarian pronunciation: [ˈbudɒpɛʃt]) is the capital and the most populous city of Hungary, and the tenth-largest city in the European Union by population within city limits.[9][10][11] The city had an estimated population of 1,752,286 in 2019 distributed over a land area of about 525 square kilometres (203 square miles).[12] Budapest is both a city and county, and forms the centre of the Budapest metropolitan area, which has an area of 7,626 square kilometres (2,944 square miles) and a population of 3,303,786, comprising 33 percent of the population of Hungary.[13][14]

https://en.wikipedia.org/wiki/Help:IPA/English
https://en.wikipedia.org/wiki/Help:IPA/Hungarian
https://en.wikipedia.org/wiki/Capital_city
https://en.wikipedia.org/wiki/List_of_cities_and_towns_of_Hungary
https://en.wikipedia.org/wiki/Hungary
https://en.wikipedia.org/wiki/Largest_cities_of_the_European_Union_by_population_within_city_limits
https://en.wikipedia.org/wiki/European_Union
https://en.wikipedia.org/wiki/Budapest#cite_note-TIME2-9
https://en.wikipedia.org/wiki/Budapest#cite_note-10
https://en.wikipedia.org/wiki/Budapest#cite_note-11
https://en.wikipedia.org/wiki/Budapest#cite_note-Encarta-12
https://en.wikipedia.org/wiki/List_of_cities_and_towns_of_Hungary
https://en.wikipedia.org/wiki/Counties_of_Hungary
https://en.wikipedia.org/wiki/Budapest_metropolitan_area
https://en.wikipedia.org/wiki/Budapest#cite_note-13
https://en.wikipedia.org/wiki/Budapest#cite_note-14

!!! Too many links !!!

→ How do we get from one page to multiple?

Automation

- repetition of code across different units

→ `for`-loops

→ `apply()` with functions

Automation

`for`- loops

```
for (i in vector) {
    code with i
}
```

`for`- loops

Your pick:
index that varies

variation
of index

Your code, generalized
to work with the
arguments

```
for (i in vector) {
  code with i
}
```

`for (i in VECTOR){ do something with i }`

`for (i in 1:2){ print(i) }`

Automation

`for`- loops

Example

```
for (i in 1:length(urls)){
  text[i] <- read_html(urls[i]) %>%
    html_elements(".text") %>%
    html_text()
}
```

Automation

`for`- loops

Advantages

- easy to write
- do not require full translation of code into functions
- easy to interrupt and continue for prolonged scraping

Disadvantages

- become inefficient for high numbers of iterations
- no swag: stackoverflow: Are For loops evil in R?

Good to know

- loops with `for` are just the most well-known type of loop
  - `while` loops, `repeat` loops, `break` and `next` clauses

Automation

Alternative: `sapply()`

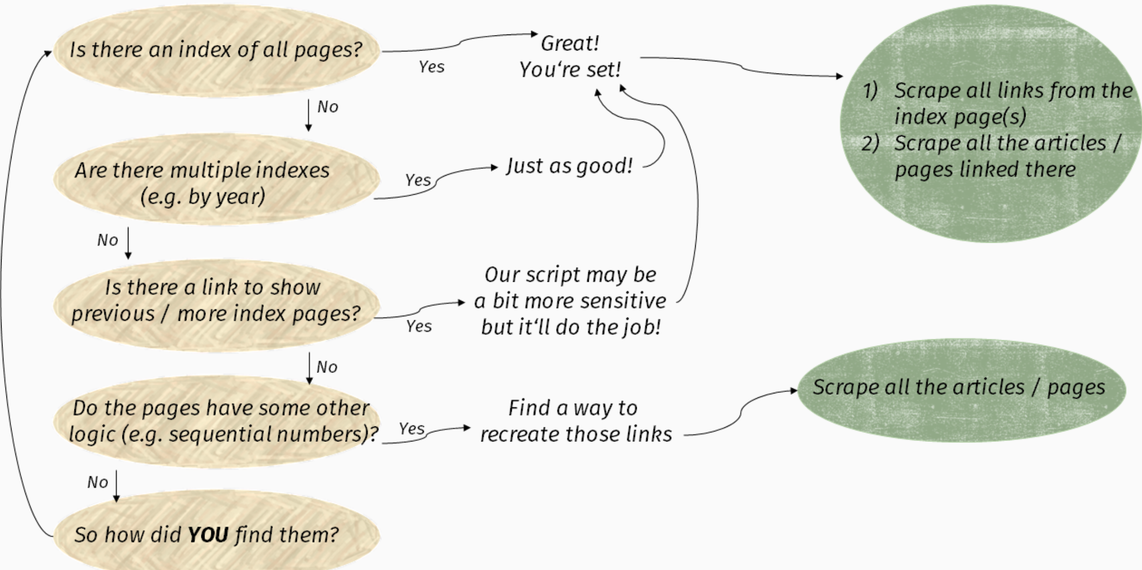
- Alternatively, we can define a **function** with scraping code

```
scrape_text <- function(url){
    page <- read_html(url)
    page %>%
    html_element(".text") %>%
    html_text()
}
```

- we use `sapply()` to apply the function to multiple URLs (see: apply commands)

```
texts <- sapply(urls,scrape_text)
```

Sequence



Scraping Scenarios

Scenarios

Scenario 1: Static pages (what we covered)

- extracting text, links and tables from many standard webpages
  - page written in HTML / XML
  - page has a static URL through which you can reach it

Procedure

- 'parsing' page in R
- extracting relevant parts
- cleaning into usable format (e.g. data frame, raw text, ...)

Scenario 2: APIs

- companies and governments often provide **application programming interfaces** for their data
  - increasing accessibility, reliability
  - used for scraping and interaction with apps

Differences to static pages

- structured data in specific notation (often JSON)
- access through sending requests
  - e.g. with `httr` package
- specified regulations on extent and volume of access
- in many cases: R packages for access to API
  - e.g. `gender`, `rtweet`, `WikipediR`, `tuber`, ...

Scenario 2: APIs

- pro
  - legal and robust to changes in webpage structure
  - highly standardized
- con
  - not every page has API
  - rate limits / restrictions to amount of data
  - may be terminated: 'post API age'

Scenario 3: dynamic pages

- for scraping pages that change while you are on them without changing their URL
  - e.g. Buzzfeed, (many) search functions, pages without permanent URL

Differences to static pages

- simulates web browsing rather than parsing static page
  - navigation & scraping through commands to automated browser
  - primarily with `RSelenium` package

- pro
  - get around many restrictions to scraping
  - possibility to automate browsing
- con
  - difficult to set up
  - less robust than static scraping

Scenario 4: web crawling / spiders

- parsing of massive amounts of data
  - e.g. price data, building a search engine, ...
- parsing of pages e.g. through `boilerpipeR`

Differences to static pages

- no selection of specific parts but use of *heuristics* on HTML code
  - → less exact but less labor-intensive extraction of content

- pro
  - masses of data
- con
  - masses of data (that are unclear)

Scenarios

Scenario 5: Task scheduling

- collecting data over time requires **regular updates**, e.g.
  - scraping daily front page news
  - updating a Corona infographic automatically with new cantonal data
- task schedulers help us to create **automatic background tasks** so we do not need to manually execute the script at regular intervals

Scheduling R tasks

- taskscheduleR or cronR - or the scheduler of your operating system
  - youtube explainer

Scenario 6: Importing offline data

- `readtext`: R package to read text from documents into R (and quanteda)
    - Documentation
- from **single files, folders, URLs, zips**
- works for, plain text files **(.txt)**, JavaScript Object Notation **(.json)**, comma-or tab-separated values **(.csv, .tab, .tsv)**, XML documents **(.xml)**, PDF **(.pdf)**, Microsoft Word formatted files **(.doc, .docx)**

```
# all word documents
texts ← readtext("*.docx")
# all pdf documents in the slides folder
docs ← readtext(file="slides/*.pdf")
# all documents from a zip
docs ← readtext(file="solutions/solutions.zip")
```

→ check the IMF example script for bulk downloading PDFs and other files!

Homework

- **06_scraping.rmd, 06_singlefile.rmd, 06_scraping_briefings.rmd**
- try `readtext` to read in some PDFs, e.g. your papers, your latest reading list, ...
- optional: dine at the CSS diner

Building on the course

- find and (if possible) gather some **text data you want to use**
  - web content
  - a dataset
  - text documents
- ...try things out!

Thank you! Questions?