

Forschungspraktikum CSS

Session 02 · Crawling the web

Dr. Sascha Grehl

2026-10-29



Outline

- What does this mean, and how do we do it?
- Introduction to HTML
- What do we need to bear in mind when scraping?

Web Scraping

- Scraping describes the process of acquiring data from the World Wide Web
- Ways of acquiring data, from most to least convenient in the source slides:
  1. **Data dumps:** companies release substantial chunks of their data
  2. **APIs (Application Programming Interfaces):** structured access to data, such as Spotify or The New York Times
  3. **Screen scraping:** a program grabs raw web content
     - Sometimes the program simulates a browser before grabbing the content

Overview of R packages for web technologies

Data dumps

Companies release substantial chunks of their data

Pro:

- Easy to acquire
- Clear legal implications

Contra:

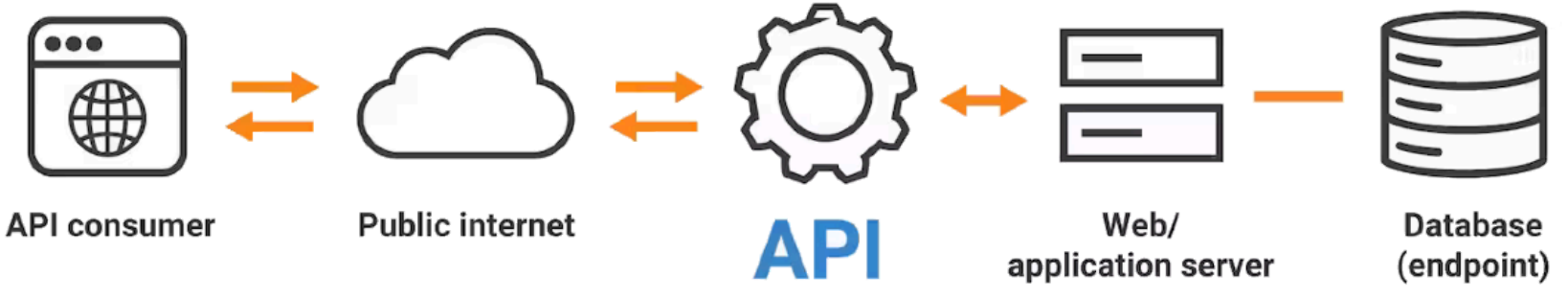
- Not always easy to process, for example SQL data
- Limited control

Examples

- Hugging Face - Datasets
- Kaggle - Datasets
- Project Gutenberg: books, accessible through the R package `gutenbergr`

APIs

- API = Application Programming Interface



How a web API works

APIs: structured requests

- Companies provide structured access to their data, such as Spotify or The New York Times

Pro

- Post-processing is usually straightforward, with clear legal implications

Contra

- Access is not always easy and depends on the documentation
  - Packages or wrappers may exist: rOpenSci overview
- APIs (sometimes) do not provide the full data

APIs: Example request

https://api.nytimes.com / svc/search/v2/articlesearch.json ? q=Trump&end_date=20161110&api-key=[your_key]

base url endpoint headers/parameters

Component	Explanation
`?`	Marks the beginning of the query string
`q`	Specifies the search term
`=`	Separates a parameter name from its value
`&`	Separates multiple query parameters
`end_date`	Restricts results to articles published
`api-key`	Provides your personal API key to authenticate the request

APIs: A JSON response

```
{
  "status": "OK",
  "response": {
    "docs": [
      { # first document
        "headline": {
          "main": "Trumpism After Trump",
          "print_headline": "Trumpism  after all the
         dust settles"
        },
        "abstact": "If the American anger and dread that
         fed Trump’s rise are not addressed, the
         consequences will be dire.",
        "pub_date": "2016-10-11T11:39:54Z",
        "section_name": "Opinion",
        "word_count": 825
      },
      { # second document ... }
    ]
} }
```

Excerpt from the NYT API response, only selected fields shown

Reading the structure

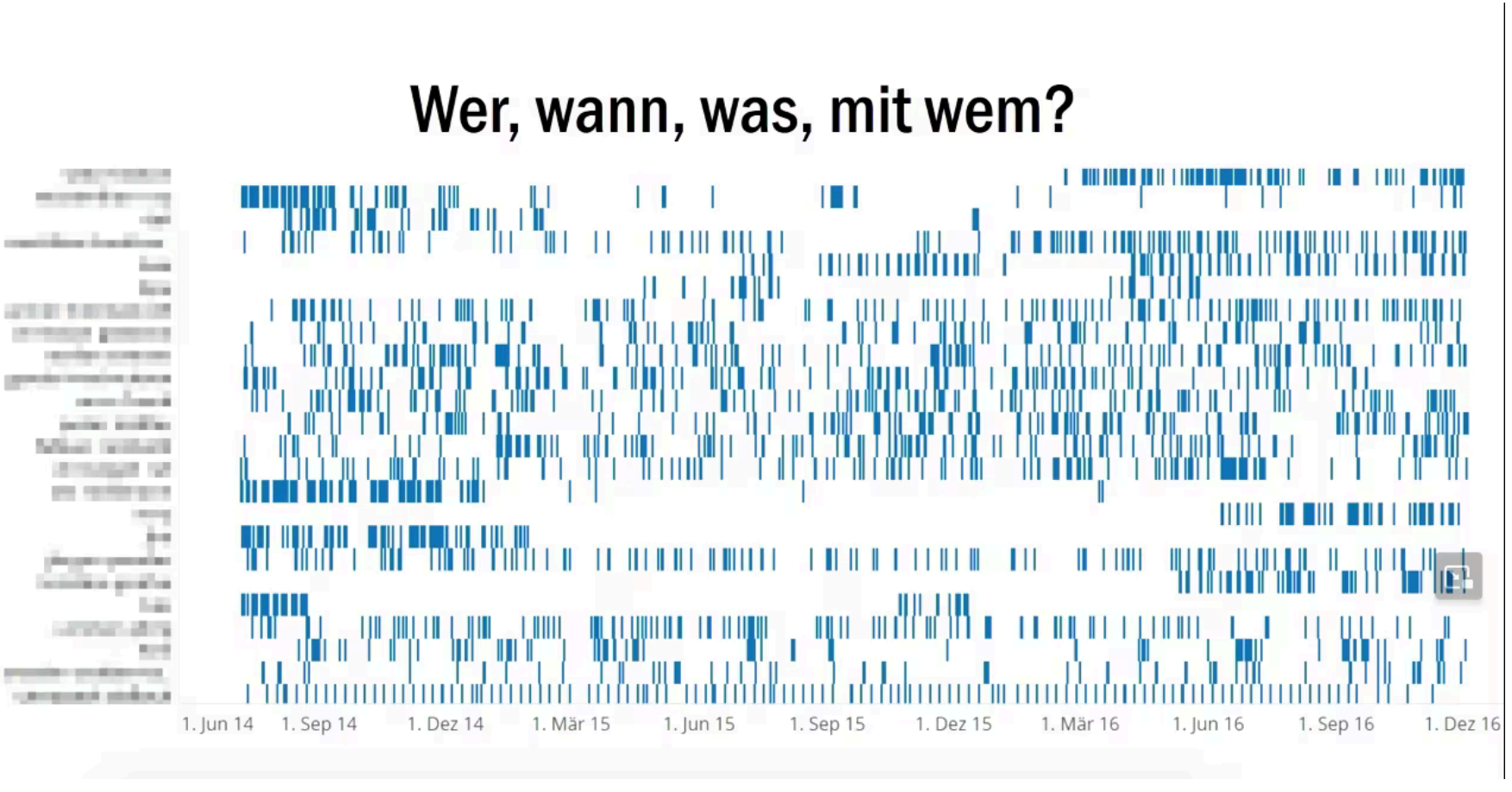
- `{ }` contains an object with named fields
- `[ ]` contains an array of items
- `response.docs` contains the articles
- Each article has nested objects and individual values

Here, the API does not provide the text of the articles!

How could this data be useful even without access to the full text of the articles?

Spiegel Online Metadata

y-axis: Authors, **x-axis**: Article publication date



David Kriesel, SpiegelMining, 33c3 Vortrag

JSON and XML

- Common formats: XML and JSON
- Extract data with `xml2` for XML documents or `jsonlite` for JSON
  - Introduction to xml2: working with XML documents is also covered in `rvest`
  - Introduction to jsonlite: transform JSON into tibbles or lists, then manipulate it with R and tidyverse functions

```
{
  "status": "OK",
  "response": {
    "docs": [
      { # first document
        "headline": {
          "main": "Trumpism After Trump",
          "print_headline": "Trumpism  after all the dust settles"
        },
        "abstract": "If the American anger and dread that fed Trump's rise are
         not addressed, the consequences will be dire.",
        "pub_date": "2016-10-11T11:39:54Z",
        "section_name": "Opinion",
        "word_count": 825
      },
      { # second document ... }
    ]
} }
```



```
Rows: 30
Columns: 20
$ abstract         <chr> "If the American anger and dread …
$ byline           <df[,1]> <data.frame[20 x 1]>
$ document_type    <chr> "article", "article", "article…
$ headline         <df[,3]> <data.frame[20 x 3]>
$ `_id`            <chr> "nyt://article/536d0ea0-c5df-5adb…
$ keywords         <list> [<data.frame[4 x 3]>], [<data.…
$ multimedia       <df[,4]> <data.frame[20 x 4]>
$ news_desk        <chr> "OpEd", "Metro", "National", "Nat…
$ print_page       <chr> "1", "1", "1", "1", "1", "1", "1…
$ print_section    <chr> "A", "A", "A", "A", "A", "A", …
$ pub_date         <chr> "2016-10-11T11:39:54Z", "2016-11-…
$ section_name     <chr> "Opinion", "New York", "U.S.", "U…
$ snippet          <chr> "If the American anger and dread …
$ source           <chr> "The New York Times", "The New Yo…
$ subsection_name  <chr> "", "", "Politics", "Politics", "…
$ type_of_material <chr> "Op-Ed", "News", "News", "News", …
$ uri              <chr> "nyt://article/536d0ea0-c5df-5adb…
$ web_url          <chr> "https://www.nytimes.com/2016/10/…
$ word_count       <int> 825, 1413, 1627, 1395, 1469, 2120…
$ search_term      <chr> "Trump", "Trump", "Trump", "Trump…
```

Workflow (and Some Ethical Considerations)

1. **Collect data**

- **Save the original data** before processing it (only raw data is *good* data)
- Avoid downloading data you have already collected (**caching**)
- Respect API rate limits and avoid unnecessary requests (API's terms of use)

2. **Validate data**

- **Continuously check for errors**, missing values, and unexpected results
- Automate these checks whenever possible
- **Log errors** and flag problematic observations
- Improve your code to handle errors and recover missing or faulty data

3. **Clean and structure data**

- *We will come to this later…*

Collect and preserve raw data

```r
for (term in search_terms) {
  # 1. Avoid unnecessary requests
  if (is_cached(term)) next

  # 2. Make the query
  response <- GET(url, query = list(q = term, ...))

  # 3. Check whether the request succeeded
  if (!is_valid(response)) {
    log_error(term, response)
    next
  }
  # 4. Preserve the original data
  save_raw(response, term)

  # 5. Be polite to the server
  Sys.sleep(10)
}
```

`is_cached()`, `is_valid()`, `log_error()`, and `save_raw()` are custom-made functions

Caching

- Always **record the date and time** of each request, either in the file name or within the data
- **Avoid downloading the same data twice** whenever possible
- However, there is one important exception to this rule

Time-Series Data

- Sometimes, we are interested in **how online data changes over time**
  - Requires downloading the same data repeatedly at different points in time
  - Each download represents a **snapshot** of the data at a specific moment
- **Never overwrite previous snapshots** - preserve them to track changes over time
- If storage space is an issue: If two snapshots are identical, we can record this in a separate database and avoid storing duplicate copies (make 100% sure they are identical!)

Example: Time-Series Overview

term	date_time	version
Trump	2026-10-20 08:00	1
Trump	2026-10-21 08:01	1
Trump	2026-10-22 08:00	1
Trump	2026-10-23 08:04	2
Trump	2026-10-24 08:00	2
Biden	2026-10-20 08:00	1
Biden	2026-10-21 08:01	1
Biden	2026-10-22 08:00	2
Biden	2026-10-23 08:06	3
Biden	2026-10-24 08:01	3

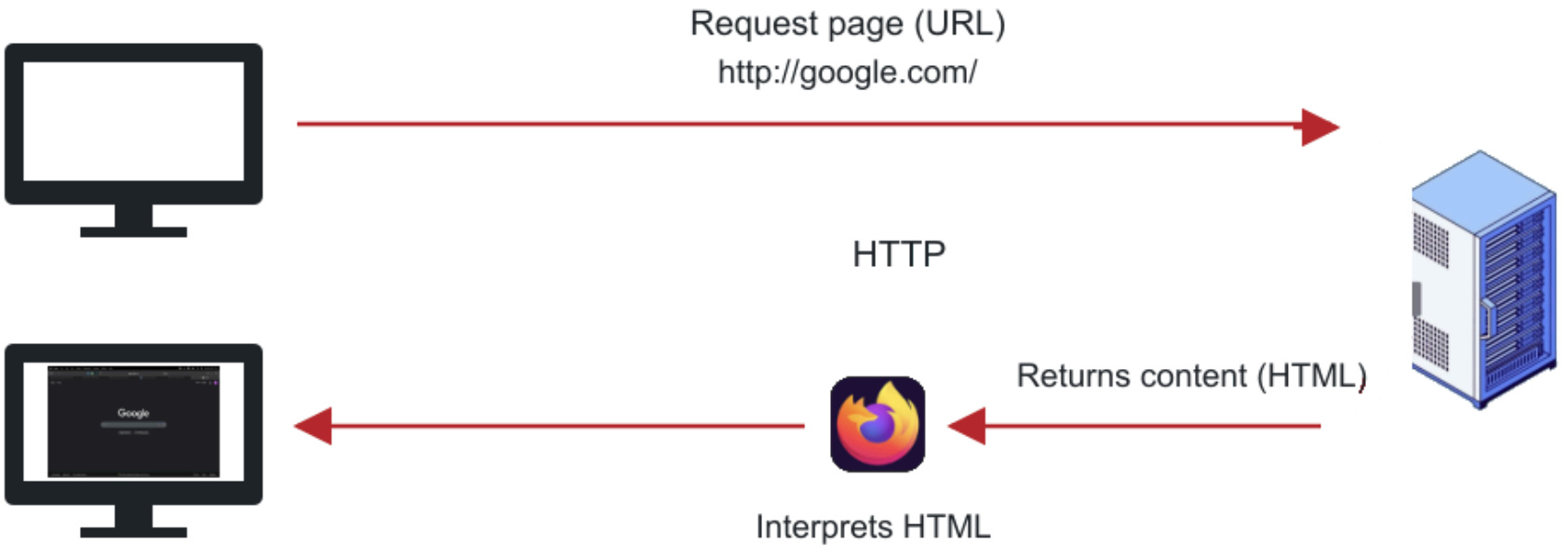
No Data Dumps, No API?

Let’s get our hands dirty … and scrape some data ourselves

Basic web terminology

- **Browser**
  - An application that interprets and displays websites
- **URL: Uniform Resource Locator**
  - The address of a web resource
- **HTTP(S): Hypertext Transfer Protocol (Secure)**
  - The protocol browsers use to communicate (securely)
- **HTML: Hypertext Markup Language**
  - The language used to structure web pages

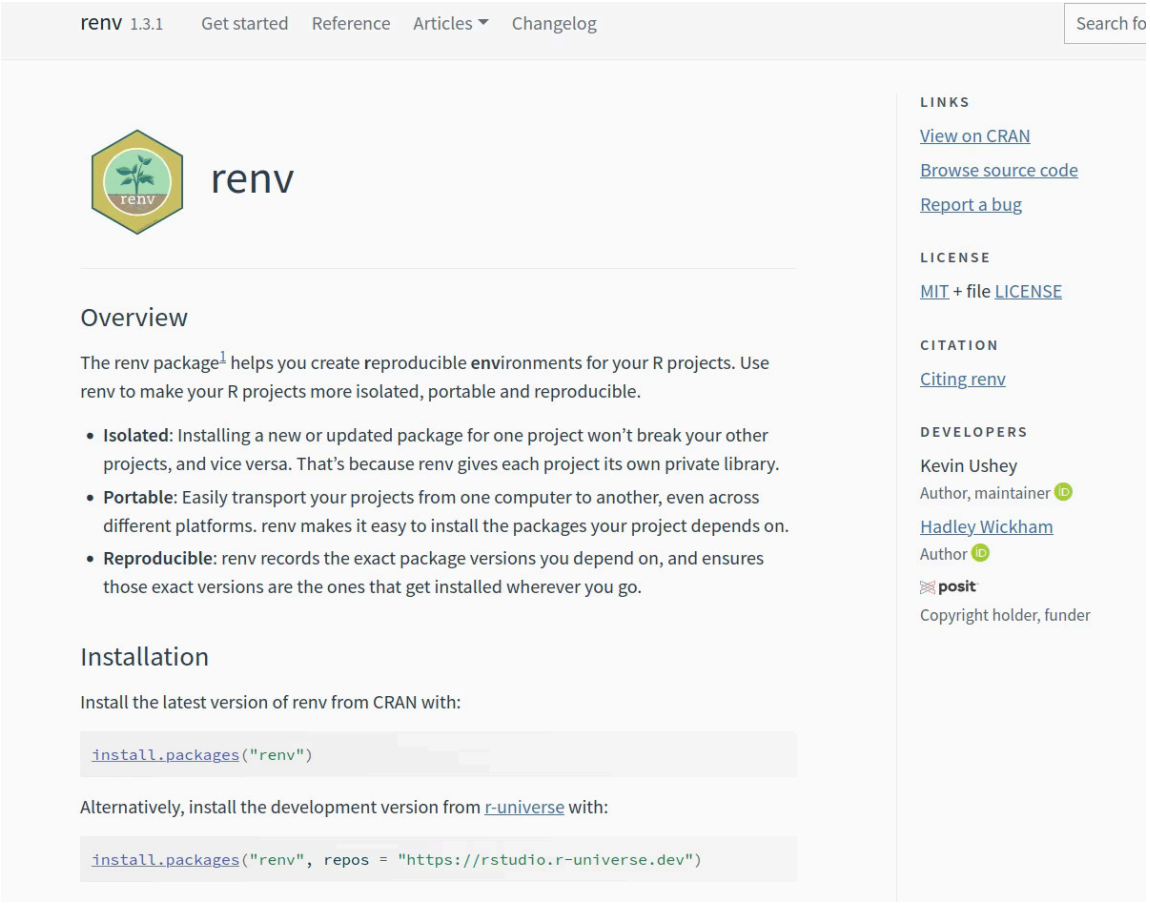
Normal browsing



Normal browsing: request above, response below, client left and server right

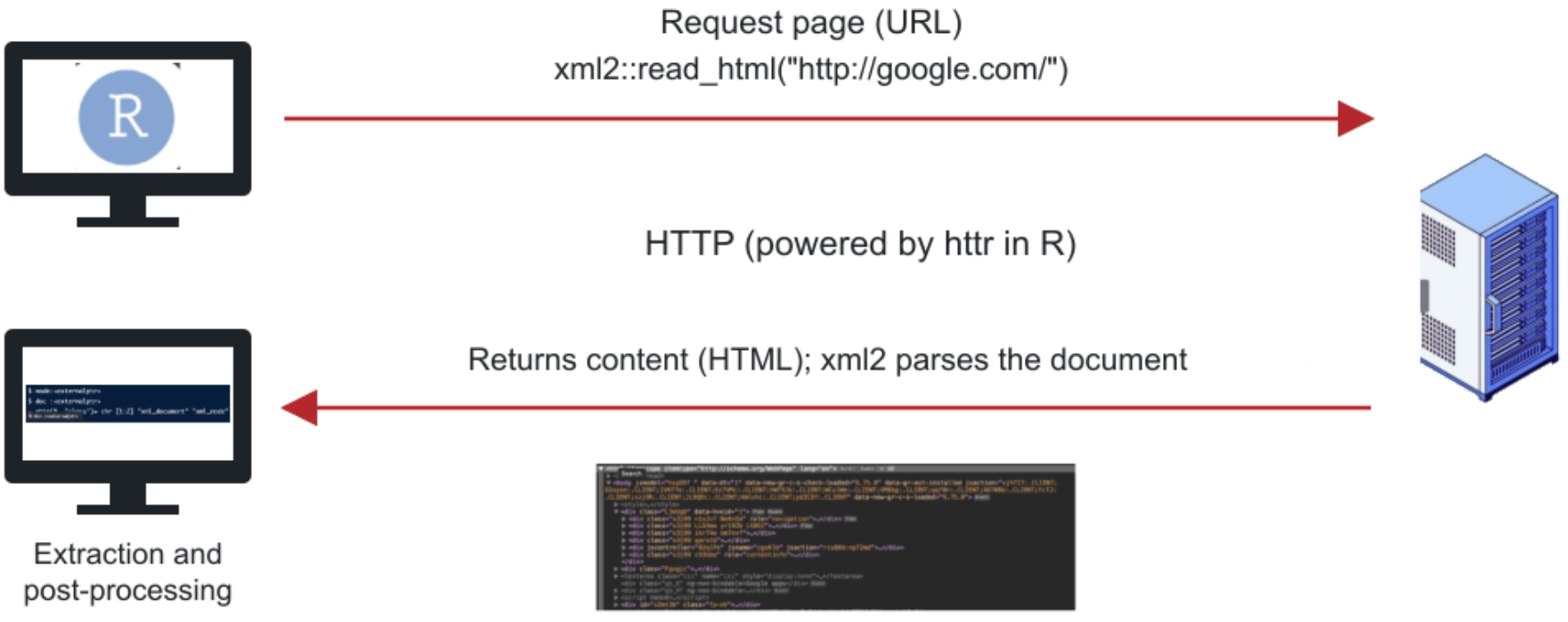
- The browser requests a page by URL, for example `http://google.com/`
- The server returns HTML over HTTP(S)
- The browser interprets and displays the HTML

Website



Assume we want to scrape the developers.

Scraping: requests and extraction



Scraping: requests and extraction: request above, response below, client left and server right

- Request the page with `read_html("http://google.com/")` (either by the `xml2` or the `rvest` package)
- The server returns HTML

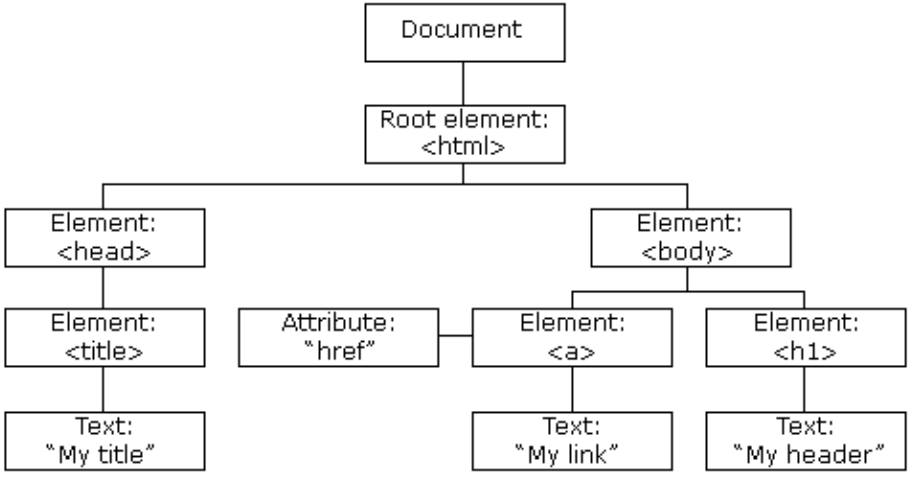
Website - As HTML

<!DOCTYPE html> <!-- Generated by pkgdown: do not edit by hand --><html lang="en"> <head> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"> <meta charset="utf-8"> <meta http-equiv="X-UA-Compatible" content="IE=edge"> <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"> <title>Project Environments • renv</title> <!-- favicons --><link rel="icon" type="image/png" sizes="96x96" href="favicon-96x96.png"> <link rel="icon" type=""image/svg+xml"" href="favicon.svg"> <link rel="apple-touch-icon" sizes="180x180" href="apple-touch-icon.png"> <link rel="icon" sizes="any" href="favicon.ico"> <link rel="manifest" href="site.webmanifest"> <script src="deps/jquery-3.6.0/jquery-3.6.0.min.js"></script><meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"> <link href="deps/bootstrap-5.3.8/bootstrap.min.css" rel="stylesheet"> <script src="deps/bootstrap-5.3.8/bootstrap.bundle.min.js"></script><link href="deps/Source_Sans_Pro-0.4.10/font.css" rel="stylesheet"> <link href="deps/Source_Code_Pro-0.4.10/font.css" rel="stylesheet"> <link href="deps/font-awesome-6.5.2/css/all.min.css" rel="stylesheet"> <link href="deps/font-awesome-6.5.2/css/v4-shims.min.css" rel="stylesheet"> <script src="deps/headroom-0.11.0/headroom.min.js"></script><script src="deps/headroom-0.11.0/jQuery.headroom.min.js"></script><script src="deps/bootstrap-toc-1.0.1/bootstrap-toc.min.js"></script><script src="deps/clipboard.js-2.0.11/clipboard.min.js"></script><script src="deps/search-1.0.0/autocomplete.jquery.min.js"></script><script src="deps/search-1.0.0/fuse.min.js"></script><script src="deps/search-1.0.0/mark.min.js"></script><!-- pkgdown --><script src="pkgdown.js"></script><link href="extra.css" rel="stylesheet"> <meta property="og:title" content="Project Environments"> <meta name="description" content="A dependency management toolkit for R. Using renv, you can create and manage project-local R libraries, save the state of these libraries to a lockfile, and later restore your library as required. Together, these tools can help make your projects more isolated, portable, and reproducible."> <meta property="og:description" content="A dependency management toolkit for R. Using renv, you can create and manage project-local R libraries, save the state of these libraries to a lockfile, and later restore your library as required. Together, these tools can help make your projects more isolated, portable, and reproducible."> <meta property="og:image" content="https://rstudio.github.io/renv/logo.svg"> <!-- Google Tag Manager --><script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start': new Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName(s)[0],j=d.createElement(s),dl=l!='dataLayer'?'&l='+l:'';j.async=true;j.src='https://www.googletagmanager.com/gtm.js?id='+i+dl;f.parentNode.insertBefore(j,f);})(window,document,'script','dataLayer','GTM-KHBDBW7');</script><!-- End Google Tag Manager --> </head> <body> Skip to content <nav class="navbar navbar-expand-lg fixed-top bg-none" data-bs-theme="light" aria-label="Site navigation"> renv <small class="nav-text text-muted me-auto" data-bs-toggle="tooltip" data-bs-placement="bottom" title="Released version">1.3.1</small> <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbar" aria-controls="navbar" aria-expanded="false" aria-label="Toggle navigation"> </button> <ul class="navbar-nav me-auto"> <li class="nav-item">Get started <li class="nav-item">Reference <li class="nav-item dropdown"> <button class="nav-link dropdown-toggle" type="button" id="dropdown-articles" data-bs-toggle="dropdown" aria-expanded="false" aria-haspopup="true">Articles</button> <ul class="dropdown-menu" aria-labelledby="dropdown-articles"> <hr class="dropdown-divider"> <h6 class="dropdown-header" data-toc-skip>Basics</h6> Package sources Installing packages Using Python with renv Frequently asked questions packrat vs. renv <hr class="dropdown-divider"> <h6 class="dropdown-header" data-toc-skip>Deployment</h6> Using renv with continuous integration Using renv with Docker Using renv with Posit Connect <hr class="dropdown-divider"> <h6 class="dropdown-header" data-toc-skip>Advanced</h6> Package development Project profiles <li class="nav-item">Changelog <ul class="navbar-nav"> <li class="nav-item"><form class="form-inline" role="search"> <input class="form-control" type="search" name="search-input" id="search-input" autocomplete="off" aria-label="Search site" placeholder="Search for" data-search-index="search.json"> </form> <li class="nav-item"> </nav> <main id="main" class="col-md-9"> <img src="logo.svg" class="logo" alt=""><h1 id="renv-">renv </h1> <!-- badges: start --> <!-- badges: end --> <h2 id="overview">Overview </h2> The renv package<a class="footnote-ref" tabindex="0" data-bs-toggle="popover" data-bs-content="<p>Pronounced “R” “env”</p>">¹ helps you create reproducible environments for your R projects. Use renv to make your R projects more isolated, portable and reproducible.   Isolated: Installing a new or updated package for one project won't break your other projects, and vice versa. That's because renv gives each project its own private library. Portable: Easily transport your projects from one computer to another, even across different platforms. renv makes it easy to install the packages your project depends on. Reproducible: renv records the exact package versions you depend on, and ensures those exact versions are the ones that get installed wherever you go.   <h2 id="installation">Installation </h2> Install the latest version of renv from CRAN with: <pre class="downlit sourceCode r"> <code class="sourceCode R">install.packages("renv")</code></pre> Alternatively, install the development version from r-universe with: <pre class="downlit sourceCode r"> <code class="sourceCode R">install.packages("renv", repos = "https://rstudio.r-universe.dev")</code></pre>   <h2 id="workflow">Workflow </h2> <img src="articles/renv.png" alt="A diagram showing the most important verbs and nouns of renv. Projects start with init(), which creates a project library using packages from the system library. snapshot() updates the lockfile using the packages installed in the project library, where restore() installs packages into the project library using the metadata from the lockfile, and status() compares the lockfile to the project library. You install and update packages from CRAN and GitHub using install() and update(), but because you'll need to do this for multiple projects, renv uses cache to make this fast." width="408" style="display: block; margin: auto;"> Use <code>renv::init()</code> to initialize renv in a new or existing project. This will set up a project library, containing all the packages you're currently using. The packages (and all the metadata needed to reinstall them) are recorded into a lockfile, <code>renv.lock</code>, and a <code>.Rprofile</code> ensures that the library is used every time you open that project. As you continue to work on your project, you will install and upgrade packages, either using <code>install.packages()</code> and <code>update.packages()</code> or <code>renv::install()</code> and <code>renv::update()</code>. After you've confirmed your code works as expected, use <code><a href="reference/snapshot.

We usually end up with something like this.

HTML basics

- We usually want only part of a page's content (even though we initially **store the complete page**)
- We need to extract the relevant information
- **Regular expressions** are one solution, but can be tedious
- HTML structure lets us select specific parts of a page
- HTML tags reference



Typical HTML structure

HTML without CSS

```
<html>
<head>
  <title>Page title</title>
</head>
<body>
  <h1>A heading</h1>
  <p>Some text &amp;amp; <b>some bold text.</b></p>
  <p>Some more text &amp;amp; <i>italicized text.</i></p>
  <p>Even more text &amp;amp; <i class="paragraph">some italicized text.</i></p>
  <p><a id="link" href="www.nyt.com">The New York Times</a></p>
</body>
</html>
```



Rendered HTML without CSS

- The page lacks styling because it has no CSS (Cascading Style Sheets)

HTML with CSS

```
<html>
<head>
<style>
h1          {color: red;}
p           {color: blue;}
p i         {color: green;}
.paragraph  {color: orange;}
#link       {
              color: black;
              font-family: courier;
            }
</style>
</head>
<body>
  <h1>A heading</h1>
  <p>Some text &amp;amp; <b>some bold text.</b></p>
  <p>Some more text &amp;amp; <i>italicized text.</i></p>
  <p>Even more text &amp;amp; <i class="paragraph">some italicized text.</i></p>
  <p><a id="link" href="www.nyt.com">The New York Times</a></p>
</body>
</html>
```



Rendered HTML with CSS

CSS and content selection

- CSS controls the styling of a page
- CSS selektors help us to extract specific content
  - Classes: .myclass or div.myclass
    - Class can be assigned to multiple elements
  - IDs: #myid
    - IDs should identify exactly one element (**In theory!**)

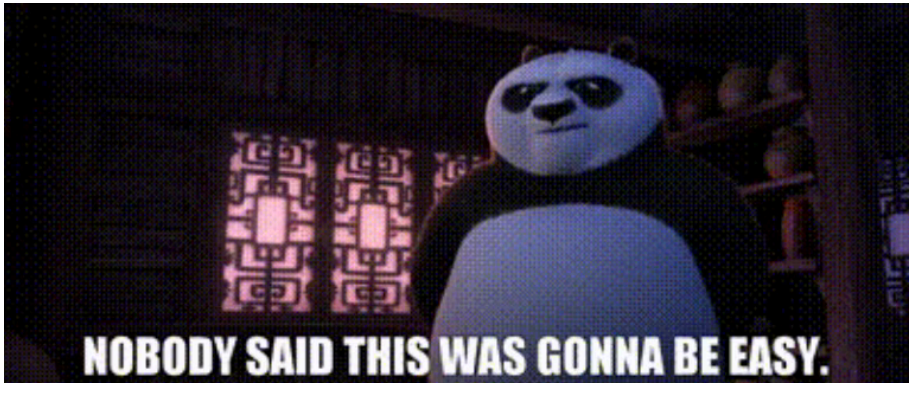
Demonstration: Inspect Element

Demonstration of the browser's **Inspect Element** tool

Extract Elements

```
page <- read_html("https://rstudio.github.io/renv/index.html")
developers_raw <- page |>
  html_element("div.developers")|>
  html_elements("li")
developers_raw
```

```
{xml_nodeset (3)}
[1] <li>Kevin Ushey <br><small class="roles"> Author, maintainer </small> <a  ...
[2] <li>\n<a href="https://hadley.nz" class="external-link">Hadley Wickham</a ...
[3] <li>\n<a href="https://www.posit.co" class="external-link"><img src="http ...
```



Extract Elements II

```
needs(rvest, purrr, tibble)

developers_df <- developers_raw |>
  map_dfr(\(li) {

    # Extract role
    role <- li |>
      html_element("small.roles") |>
      html_text2()

    # Extract name from the text before <br>
    name <- li |>
      html_text2() |>
      strsplit("\n") |>
      unlist() |>
      head(1) |>
      trimws()

    # Handle image-only entries
    if (name == "" || is.na(name)) {
      name <- li |>
        html_element("img") |>
        html_attr("alt")
    }

    tibble(
      name = name,
      role = role
    )
  })

developers_df
```

```
# A tibble: 3 × 2
  name           role
  <chr>          <chr>
1 Kevin Ushey    Author, maintainer
2 Hadley Wickham Author
3 Posit          Copyright holder, funder
```

SelectorGadget

- Finding the right selectors can be hard
- Luckily, tools such as SelectorGadget exist

Demonstration: SelectorGadget

Demonstration of SelectorGadget

Beyond static sites

- Not all websites are basic static pages (e.g. quotes.toscrape.com/js)
- Challenges include
  - Dynamic content generated with JavaScript
  - Authentication
  - CAPTCHAs that check whether a human is using the website
- These challenges may exceed the capabilities of `rvest`

```
1 read_html("https://quotes.toscrape.com/js") |>
2   html_elements(".text")
```

```
{xml_nodeset (0)}
```

Next time

- Ethics
- `selenium` lets us control a browser through code
- Implementation in R and Python

Merci!

sascha.grehl@uni-leipzig.de

Universität Leipzig

References and further practice

- Tidyverse Style Guide: files, syntax, functions and pipes
- Tidy Data: observations and variables
- purrr: map and walk: iteration
- here: declare the project root: portable paths
- renv introduction: package environments
- stringr regex reference

Running a simple loop

```r
needs::needs(tidyverse, httr, jsonlite, purrr)
url <- "https://api.nytimes.com/svc/search/v2/articlesearch.json"
search_terms <- c("Trump", "Biden", "Obama")

articles <- map_dfr(search_terms, \(term) {
  response <- GET(
    url,
    query = list(
      q = term,
      end_date = "20161110",
      `api-key` = Sys.getenv("NYT_API_KEY")
    )
  )

  result <- fromJSON(
    content(response, as = "text", encoding = "UTF-8")
  )$response$docs

  Sys.sleep(4)
  as_tibble(result) |>
    mutate(search_term = term)
})
```