```
library(dplyr)
rladies_global %>%
  filter(city == 'Brussels')
```

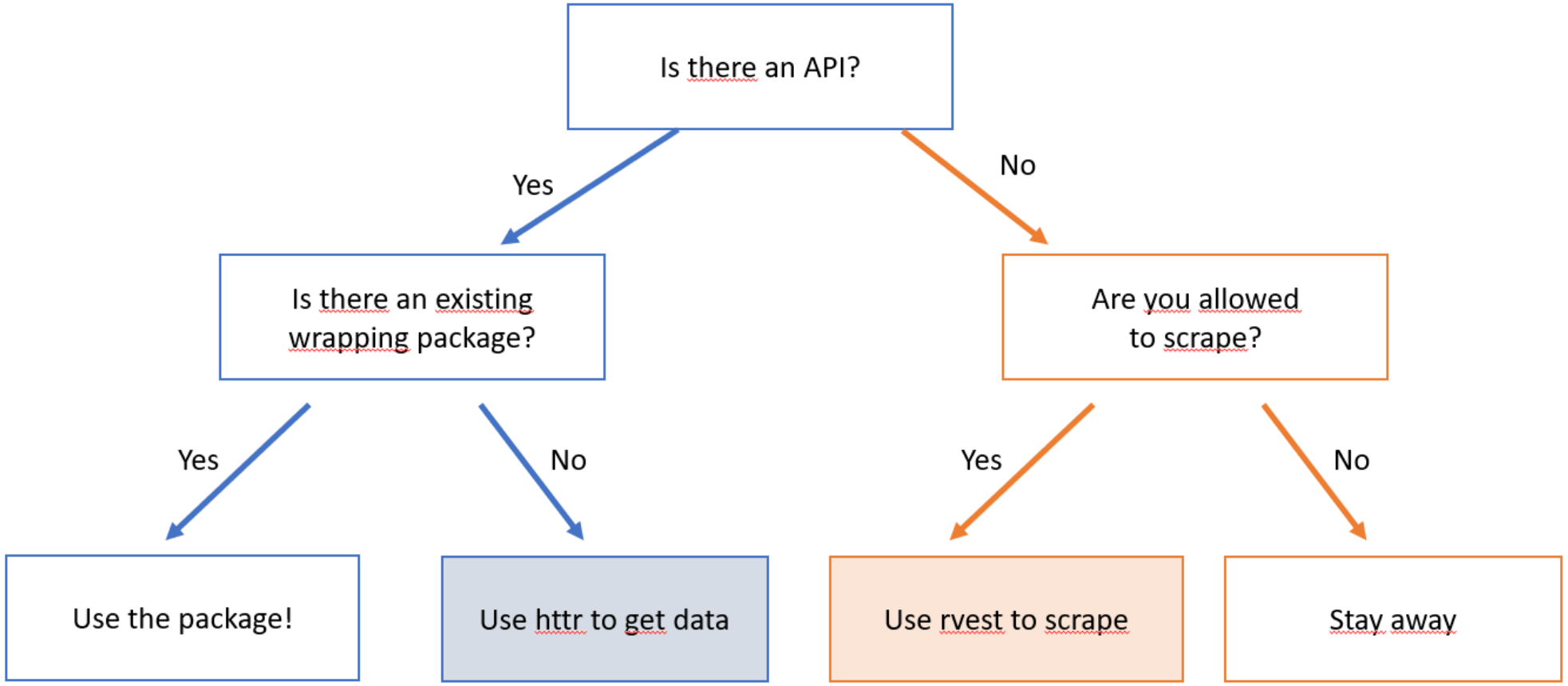
API and webscraping:

getting data from the web straight into R

Getting data from the web

- Downloading files
- Getting data from API's
- Scraping from webpages

Getting data from the web



Webscraping Etiquette

1. If there is a public API available, use the API and don't scrape the info
2. Always check with `robotstxt` or `polite` whether you are allowed to scrape a page
3. Build system sleeps into your API or scraping requests
4. Don't gather data you won't need
5. Keep away from gathering PII (personal identifiable) data

Part 1: API


esc
F1
F2
F3
F4
F5
F6
~
`
!
1
@
2
#
3
$
4
%
5
^
6
tab
Q
W
E
R
T
caps lock
A
S
D
F
G
shift
Z
X
C
V
fn
control
alt
option
command

What is an API?

API = Application Programming Interface
"a set of subroutine definitions, communication protocols,
and tools for building software"

What is an API?

API = Application Programming Interface
"a set of subroutine definitions, communication protocols,
and tools for building software"

HUH ???

What is an API?

API = Application Programming Interface
"a set of subroutine definitions, communication protocols,
and tools for building software"

HUH ???

Examples:

- Travel websites getting info on flight schedule and prices
- Getting fonts for your website (or these slides) through Google Fonts API
- Getting tweets with the hashtag #rstats via Twitter API
- ...

API communication

Communication with HTTP request methods.
Two most common protocols:

API communication

Communication with HTTP request methods.
Two most common protocols:

GET requests

Asking the API for something

```
library(httr)
response <- GET(url)
content(response)
```

API communication

Communication with HTTP request methods.
Two most common protocols:

GET requests

Asking the API for something

```
library(httr)
response <- GET(url)
content(response)
```

POST requests

Giving something to the API

```
library(httr)
response <- POST(url, body)
content(response)
```

API output

- Two general formats for API responses
- Both are plain text formats to exchange data

JSON: Javascript Object Notation Most common these days.

```
{"employees":[
    {"firstName":"John", "lastName":"Do
    {"firstName":"Anna", "lastName":"Sm
    {"firstName":"Peter", "lastName":"J
]}
```

API output

- Two general formats for API responses
- Both are plain text formats to exchange data

JSON: Javascript Object Notation Most common these days.

```
{"employees":[
    {"firstName":"John", "lastName":"Do
    {"firstName":"Anna", "lastName":"Sm
    {"firstName":"Peter", "lastName":"J
]}
```

XML: Extensible Markup Language

```
<employees>
    <employee>
        <firstName>John</firstName>
        <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName>
        <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName>
        <lastName>Jones</lastName>
    </employee>
```

A quick intro to JSON

Two basic forms:

- JSON object: `{"key": "value"}`
- JSON array: `[1, 2, 3]` or `["a", "b", "c"]`

Objects and arrays can be nested to build hierarchies.

A quick intro to JSON

Two basic forms:

- JSON object: `{"key": "value"}`
- JSON array: `[1, 2, 3]` or `["a", "b", "c"]`

Objects and arrays can be nested to build hierarchies.

For R users:

- JSON object ~ named value
- JSON array ~ named vector

A quick intro to JSON: making JSON human readable

```
library(jsonlite)
json_object <- '{"organization" : "R-Ladies", "chapter": "Brussels", "start_year": 2
prettify(json_object)
```

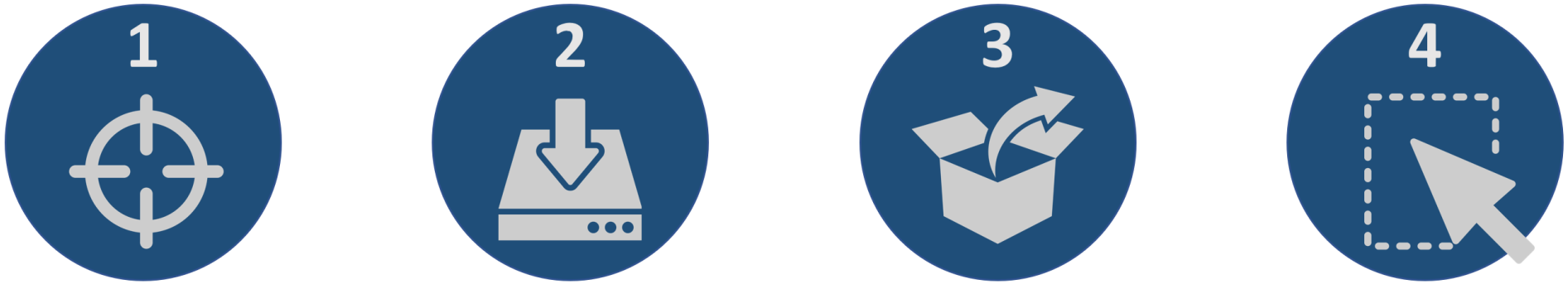
```
## {
##     "organization": "R-Ladies",
##     "chapter": "Brussels",
##     "start_year": 2017,
##     "active": true,
##     "organizers": [
##         "Marlene",
##         "Huong",
##         "Suzan"
##     ]
## }
##
```

A quick intro to JSON: parsing JSON to R

```
library(jsonlite)
fromJSON(json_object)
```

```
## $organization
## [1] "R-Ladies"
##
## $chapter
## [1] "Brussels"
##
## $start_year
## [1] 2017
##
## $active
## [1] TRUE
##
## $organizers
## [1] "Marlene" "Huong"   "Suzan"
```

API workflow



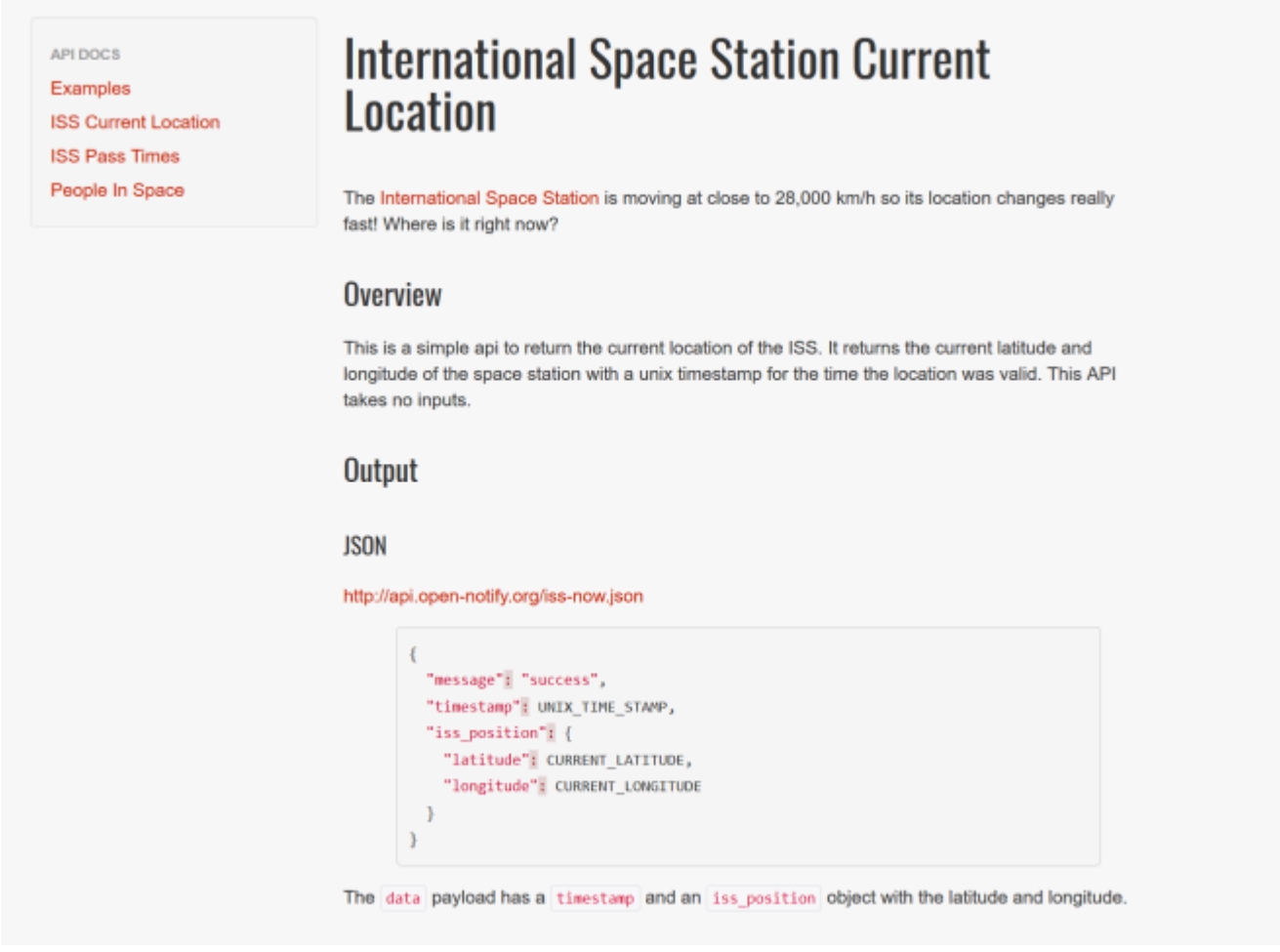
Prepare
- URL construction
- Authentication

GET request

Extract
and parse content

Select
the info you need

Easy example: ISS position



API DOCS

Examples

ISS Current Location

ISS Pass Times

People In Space

International Space Station Current Location

The International Space Station is moving at close to 28,000 km/h so its location changes really fast! Where is it right now?

Overview

This is a simple api to return the current location of the ISS. It returns the current latitude and longitude of the space station with a unix timestamp for the time the location was valid. This API takes no inputs.

Output

JSON

http://api.open-notify.org/iss-now.json

```
{
  "message": "success",
  "timestamp": UNIX_TIME_STAMP,
  "iss_position": {
    "latitude": CURRENT_LATITUDE,
    "longitude": CURRENT_LONGITUDE
  }
}
```

The `data` payload has a `timestamp` and an `iss_position` object with the latitude and longitude.

Easy example: ISS position

Step 1: Preparation:

```
url <- "http://api.open-notify.org/iss-now.json"
```

Easy example: ISS position

Step 2: GET request: Watch the status code!

```
library(httr)
response_iss <- GET(url)
response_iss
```

```
## Response [http://api.open-notify.org/iss-now.json]
##   Date: 2018-09-27 13:42
##   Status: 200
##   Content-Type: application/json
##   Size: 114 B
```

Easy example: ISS position

Step 3: Extract and parse the response

```
content_iss <- content(response_iss)
content_iss
```

```
## $message
## [1] "success"
##
## $timestamp
## [1] 1538055746
##
## $iss_position
## $iss_position$latitude
## [1] "-47.4611"
##
## $iss_position$longitude
## [1] "-43.0891"
```

Easy example: ISS position

Step 4: Data wrangling in R

```
ISS_position <- data.frame(
  time = content_iss$timestamp,
  lat = content_iss$iss_position$latitude,
  long = content_iss$iss_position$longitude)

ISS_position
```

```
##         time      lat     long
## 1 1538055746 -47.4611 -43.0891
```

API workflow

Prepare

- URL construction
- Authentication

`paste0()`
`httr::modify_url()`

`httr::authenticate()`
`httr::oauth_app()`

GET request

`httr::GET()`

Extract

and parse content

`httr::content()`

Select

the info you need

list functions:
- `purrr` package
- `rlist`

Prepare your request: URL-building

Read the documentation: every API is different

- Directory based URLs: `url/api/directory1/directory2`
- Parameter based URLs: `url/api?param1=value1&param2=value2`
- A combination of both: `url/api/directory1?param1=value`

Prepare your request: URL-building

Read the documentation: every API is different

- Directory based URLs: `url/api/directory1/directory2`
- Parameter based URLs: `url/api?param1=value1&param2=value2`
- A combination of both: `url/api/directory1?param1=value`

Examples:

Github: `https://api.github.com/repos/vmg/redcarpet/issues?state=closed`

Star Wars API: `https://swapi.co/api/planets`

Game of Thrones API: `https://www.anapioficeandfire.com/api/characters?page=1&pageSize=10`

Prepare your request: URL-building

- Option 1: Manual copy paste:

```
url <- "http://api.website.com/directory1?param1=value1&param2=value2"
```

Prepare your request: URL-building

- Option 1: Manual copy paste:

```
url <- "http://api.website.com/directory1?param1=value1&param2=value2"
```

- Option 2: Pasting together:

```
url1 <- paste("http://api.website.com", directory1, sep = "/")

url2 <- paste0("http://api.website.com/directory1?param1=", value1,
               "&param2=", value2)
```

Prepare your request: URL-building

- Option 1: Manual copy paste:

```
url <- "http://api.website.com/directory1?param1=value1&param2=value2"
```

- Option 2: Pasting together:

```
url1 <- paste("http://api.website.com", directory1, sep = "/")

url2 <- paste0("http://api.website.com/directory1?param1=", value1,
               "&param2=", value2)
```

- Option 3: Modifying url

```
url <- modify_url(url = "http://api.website.com",
                  path = "directory1",
                  query = list(param1 = "value1", param2 = "value2"))
```

Prepare your request: Authentication

Different types of authentication exists:

- username/password via `authenticate()`
- OAuth 1.0 via `oauth_app()` and `oauth1.0_token`
- OAuth 2.0 via `oauth_app()` and `oauth2.0_token`

How to get access?

- Follow the documentation
- Follow demo's: https://github.com/r-lib/httr/tree/master/demo

Step 2: GET request

Basic structure:

```
library(httr)
response <- GET(url,
                use_agent = "mycontact@email.com",
                query = query_params,
                authenticate("username", "password"))
```

- URL: base URL or full URL assembled before
- optional but recommended: add a user-agent to describe who you are and what you're trying to do
- optional: query with a list of query paramaters if not added to url yet
- optional: authenticate

Step 3: Extract and parse the response


```
library(httr)
content <- content(response)
```

By default, httr::content() automatically parses to R

- if JSON: using jsonlite::fromJSON()
- if XML: using xml2::read_xml()

Alternative options:

- `content(response, as = "text")`: extracts without parsing

Step 4: Data wranging in R

Use all your list tools to get your data in the shape you need it:

- purrr
- dplyr
- rlist
- ...

Repeated API calls

- Wrap your code in a function
- Build a Sys.sleep() between iterations!

Your turn!

Your turn: Example 1

Get the five next ISS passes above Brussels

API Documentation:

http://open-notify.org/Open-Notify-API/ISS-Pass-Times/

Coordinates brussels

- lat_bxl <- 50.85045
- long_bxl <- 4.34878

Your turn: solution example 1

```
lat_bxl <- 50.85045
long_bxl <- 4.34878

#option 1: manual
url <- "http://api.open-notify.org/iss-pass.json?lat=50.85045&lon=4.34878"

#option 2: paste together
url2 <- paste0("http://api.open-notify.org/iss-pass.json?lat=", lat_bxl, "&lon=", lo

#option 3: build URL
url3 <- httr::modify_url(url = "http://api.open-notify.org",
                         path = "iss-pass.json",
                         query = list(lat = lat_bxl, lon = long_bxl))
```

Your turn: solution example 1

```
#GET request
response <- GET(url3)

#unpack response
content <- content(response)

#modify to R
pass_times <- data.frame(
  risetime = purrr::map_chr(content$response, "risetime"),
  duration = purrr::map_chr(content$response, "duration"))

pass_times
```

```
##     risetime duration
## 1 1538852887      628
## 2 1538858707      479
## 3 1538913262      299
## 4 1538918841      597
## 5 1538924587      645
```

Your turn: Example 2

An API of Ice and Fire: www.anapioficeandfire.com

- Get the info on all the books
- Find all book names
- Extract the POV (point of view characters) from book 1
- Ask the API for info on all pov characters from book 1

Your turn: solution example 2

Getting the info on all books

```
library(httr)
url <- "https://www.anapioficeandfire.com/api/books"
response_books <- GET(url)
content_books <- content(response_books)

#look into the book
str(content_books, max.level = 2)
```

```
## List of 10
##  $ :List of 11
##   ..$ url          : chr "https://www.anapioficeandfire.com/api/books/1"
##   ..$ name         : chr "A Game of Thrones"
##   ..$ isbn         : chr "978-0553103540"
##   ..$ authors      :List of 1
##   ..$ numberOfPages: int 694
##   ..$ publisher    : chr "Bantam Books"
##   ..$ country      : chr "United States"
```

Your turn: solution example 2

Find all book names

```
purrr::map_chr(content_books, "name")
```

```
##  [1] "A Game of Thrones"          "A Clash of Kings"
##  [3] "A Storm of Swords"          "The Hedge Knight"
##  [5] "A Feast for Crows"          "The Sworn Sword"
##  [7] "The Mystery Knight"         "A Dance with Dragons"
##  [9] "The Princess and the Queen" "The Rogue Prince"
```

Your turn: solution example 2

Extract the POV (point of view characters) from book 1

```
content_book1 <- content_books[[1]]
book1_pov <- content_book1$povCharacters
book1_pov <- unlist(book1_pov)
book1_pov
```

```
## [1] "https://www.anapioficeandfire.com/api/characters/148"
## [2] "https://www.anapioficeandfire.com/api/characters/208"
## [3] "https://www.anapioficeandfire.com/api/characters/232"
## [4] "https://www.anapioficeandfire.com/api/characters/339"
## [5] "https://www.anapioficeandfire.com/api/characters/583"
## [6] "https://www.anapioficeandfire.com/api/characters/957"
## [7] "https://www.anapioficeandfire.com/api/characters/1052"
## [8] "https://www.anapioficeandfire.com/api/characters/1109"
## [9] "https://www.anapioficeandfire.com/api/characters/1303"
```

Your turn: solution example 2

Call the API for info on first character

```
url1 <- book1_pov[1]
response <- GET(url1)
content <- content(response)

data.frame(name = content$name,
  gender = content$gender,
  culture = content$culture,
  born = content$born,
  died = content$died)
```

```
##         name gender  culture                      born died
## 1 Arya Stark Female Northmen In 289 AC, at Winterfell
```

Your turn: solution example 2

Wrap it inside a function:

```
get_pov_info <- function(url) {
  response <- GET(url)
  content <- content(response)

  df <- data.frame(
    name = content$name,
    gender = content$gender,
    culture = content$culture,
    born = content$born,
    died = content$died, stringsAsFactors = FALSE)

  Sys.sleep(1)

  df
}
```

Your turn: solution example 2

```
all_pov_info <- purrr::map_df(book1_pov, get_pov_info)
all_pov_info
```

```
##                 name gender  culture                        born
## 1         Arya Stark Female Northmen    In 289 AC, at Winterfell
## 2      Brandon Stark   Male Northmen    In 290 AC, at Winterfell
## 3      Catelyn Stark Female Rivermen      In 264 AC, at Riverrun
## 4       Eddard Stark   Male Northmen    In 263 AC, at Winterfell
## 5           Jon Snow   Male Northmen                 In 283 AC
## 6        Sansa Stark Female Northmen    In 286 AC, at Winterfell
## 7   Tyrion Lannister   Male          In 273 AC, at Casterly Rock
## 8               Will   Male
## 9 Daenerys Targaryen Female Valyrian   In 284 AC, at Dragonstone
##                                                      died
## 1
## 2
## 3                                In 299 AC, at the Twins
## 4 In 299 AC, at Great Sept of Baelor in King's Landing
## 5
```

Final word in APIs...

Make you life easy:

If a package already exists: use it!

Part 2: Webscraping

Webscraping workflow

Prepare
- URL construction
- Allowed to scrape?

Load info
get a local version

Select info
grab the node(s)

Extract
and parse to R

Basics of HTML

- HTML = Hypertext Markup Language
- Describes the structure of Web pages
- Consists of *elements*:

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

Basics of HTML

- HTML = Hypertext Markup Language
- Describes the structure of Web pages
- Consists of *elements*:

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

- Elements can have 1 or more attributes that give additional information:

```
<tag attribute_name = "attribute_value"> content </tag>
```

Basics of CSS selectors

- CSS = Cascading Style Sheets
- Describe how the HTML structure should really look like (colors/fonts/...)
- CSS selectors are very useful in webscraping

Basics of CSS selectors

- CSS = Cascading Style Sheets
- Describe how the HTML structure should really look like (colors/fonts/...)
- CSS selectors are very useful in webscraping

```
.intro {
    color: #1F608B;
    font-size: 18px;
    font-weight: bold; }
```

These classes come back inside HTML:

```
<div class = "intro"> content </div>
```

Basics of CSS selectors

How do you find out what CSS selectors you can use?

- selectorgadget
- Firefox/chrome: Ctrl+Shift+I then Ctrl+Shift+C to hover over parts of the site

Webscraping workflow

Prepare

- URL construction
- Allowed to scrape?

Load info

get a local version

Select info

grab the node(s)

Extract

and parse to R

Scraping workflow: Preparation

Step 1a: URL construction:

same as before!

Scraping workflow: Preparation

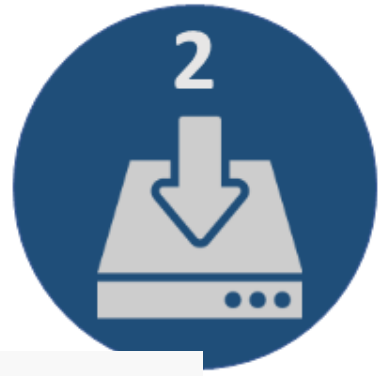
Step 1a: URL construction:

same as before!

Step 1b: Are you allowed to scrape?

```
robotstxt::paths_allowed(url)
```

Step 2: Get a local version in R



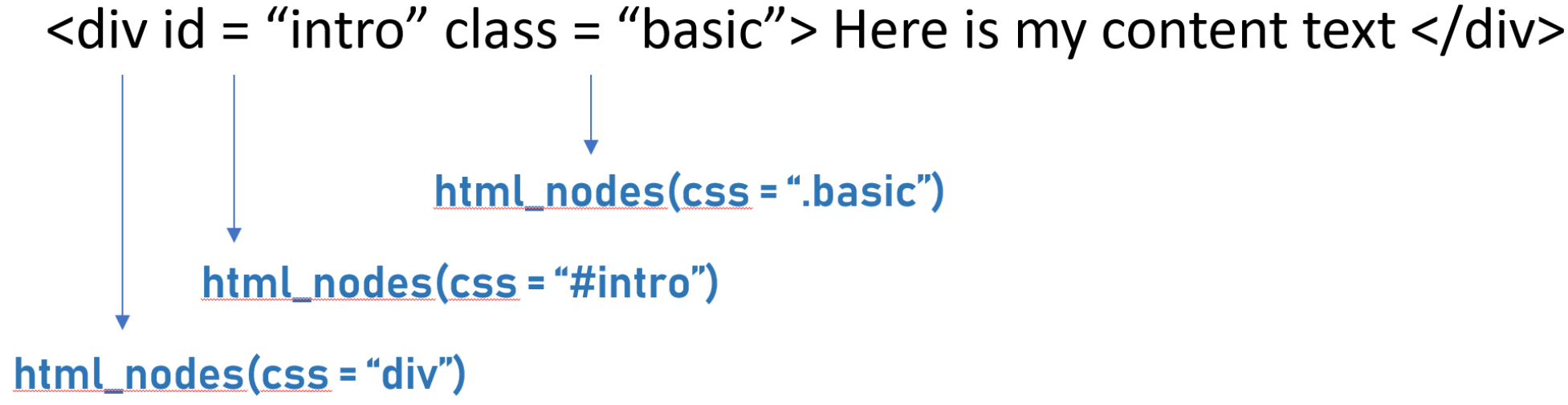
The function `read_html` will return a local XML copy of the site.

```
library(rvest)
local_copy <- read_html(url)
```

Step 3: Selecting the nodes you want

Selecting the parts you want with css selectors:

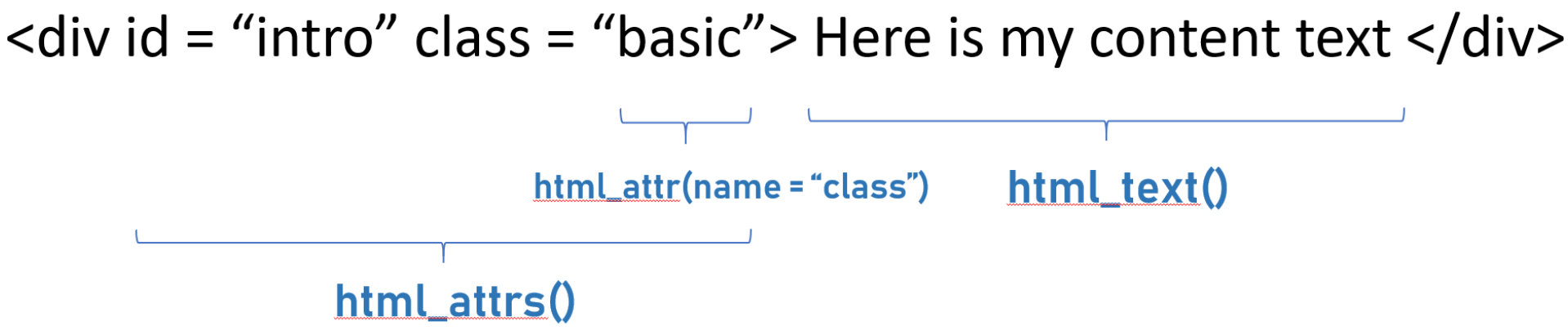
- via tagname: `css = "tagname"`
- via CSS class: `css = ".classname"`
- via CSS ID: `css = "#IDname"`



Step 4: Parse data into R

The function `html_nodes` returned an XML element. To get to content:

- parse a table: `html_table()`
- parse the content between tags: `html_text()`
- parse all attribute content: `html_attrs()`
- parse the content of a specific ttribute: `html_attr(name = "attribute_name")`



Webscraping workflow

Prepare

- URL construction
- Allowed to scrape?

`paste0()`
`httr::modify_url()`

`robotstxt::paths_allowed()`

Load info

get a local version

`rvest::read_html()`

Select info

grab the node(s)

`rvest::html_node()`
`rvest::html_nodes()`

Extract

and parse to R

`rvest::html_text()`
`Rvest::html_attr()`
`rvest::html_table()`

Demo time

Your turn

Your turn: exercise 1

URL: https://en.wikipedia.org/wiki/R_(programming_language)
Task: get this table into R

Milestones [edit]

A list of changes in R releases is maintained in various "news" files at CRAN.[45] Some highlights are listed below for several major releases.

Release	Date	Description
0.16		This is the last alpha version developed primarily by Ihaka and Gentleman. Much of the basic functionality from the "White Book" (see S history) was implemented. The mailing lists commenced on April 1, 1997.
0.49	1997-04-23	This is the oldest source release which is currently available on CRAN.[46] CRAN is started on this date, with 3 mirrors that initially hosted 12 packages.[47] Alpha versions of R for Microsoft Windows and the classic Mac OS are made available shortly after this version.[*citation needed*]
0.60	1997-12-05	R becomes an official part of the GNU Project. The code is hosted and maintained on CVS.
0.65.1	1999-10-07	First versions of update.packages and install.packages functions for downloading and installing packages from CRAN.[48]
1.0	2000-02-29	Considered by its developers stable enough for production use.[49]
1.4	2001-12-19	S4 methods are introduced and the first version for Mac OS X is made available soon after.
2.0	2004-10-04	Introduced lazy loading, which enables fast loading of data with minimal expense of system memory.
2.1	2005-04-18	Support for UTF-8 encoding, and the beginnings of internationalization and localization for different languages.
2.11	2010-04-22	Support for Windows 64 bit systems.
2.13	2011-04-14	Adding a new compiler function that allows speeding up functions by converting them to byte-code.
2.14	2011-10-31	Added mandatory namespaces for packages. Added a new parallel package.
2.15	2012-03-30	New load balancing functions. Improved serialisation speed for long vectors.
3.0	2013-04-03	Support for numeric index values 2^{31} and larger on 64 bit systems.
3.4	2017-04-21	Just-in-time compilation (JIT) of functions and loops to byte-code enabled by default.
3.5	2018-04-23	Packages byte-compiled on installation by default. Compact internal representation of integer sequences. Added a new serialisation format to support compact internal representations.

Solution: exercise 1

```
library(rvest)
wiki_url <- "https://en.wikipedia.org/wiki/R_(programming_language)"
robotstxt::paths_allowed(wiki_url)
```

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

Solution: exercise 1

```
local_wiki <- read_html(wiki_url)

local_wiki %>%
  html_nodes(css = ".wikitable") %>%
  html_table()
```

```
## [[1]]
##    Release       Date
## 1     0.16
## 2     0.49 1997-04-23
## 3     0.60 1997-12-05
## 4   0.65.1 1999-10-07
## 5      1.0 2000-02-29
## 6      1.4 2001-12-19
## 7      2.0 2004-10-04
## 8      2.1 2005-04-18
## 9     2.11 2010-04-22
## 10    2.13 2011-04-14
## 11    2.14 2011-10-31
```

Your turn: exercise 2

Make a dataframe with the number of participants for 10 Datacamp R courses

- Scrape links to all R courses
- Make a function to scrape participants from a page
- Iterate over 10 different courses (!!Sys.sleep!!)

Solution: exercise 2

Getting all the links to all R courses:

```
library(rvest)
library(dplyr)

course_overview_url <- "https://www.datacamp.com/courses/tech:r"
local_copy <- read_html(course_overview_url)

all_course_links <- local_copy %>%
  html_nodes(css = ".course-block > a") %>%
  html_attr(name = "href")
```

Solution: exercise 2

Getting all the links to all R courses:

```
all_course_links
```

```
##   [1] "/courses/free-introduction-to-r"
##   [2] "/courses/data-table-data-manipulation-r-tutorial"
##   [3] "/courses/dplyr-data-manipulation-r-tutorial"
##   [4] "/courses/ggvis-data-visualization-r-tutorial"
##   [5] "/courses/reporting-with-r-markdown"
##   [6] "/courses/intermediate-r"
##   [7] "/courses/introduction-to-machine-learning-with-r"
##   [8] "/courses/cleaning-data-in-r"
##   [9] "/courses/intermediate-r-practice"
##  [10] "/courses/data-visualization-with-ggplot2-1"
##  [11] "/courses/data-visualization-with-ggplot2-2"
##  [12] "/courses/data-visualization-with-ggplot2-part-3"
##  [13] "/courses/intro-to-text-mining-bag-of-words"
##  [14] "/courses/exploring-pitch-data-with-r"
##  [15] "/courses/working-with-the-rstudio-ide-part-1"
```

Solution: exercise 2

Getting the participants for one page

```
course_link <- all_course_links[1]
url <- httr::modify_url("https://www.datacamp.com", path = course_link)

robotstxt::paths_allowed(url)
```

```
##
 www.datacamp.com

## [1] TRUE
```

Solution: exercise 2

Getting the participants for one page

```
local_copy_course <- read_html(url)

local_copy_course %>%
  html_nodes(css = ".header-hero__stat--participants") %>%
  html_text() %>%
  readr::parse_number()
```

```
## [1] 962611
```

Solution: exercise 2

```
get_participants <- function(course_link) {
  url <- httr::modify_url("https://www.datacamp.com", path = course_link)
  allowed <- robotstxt::paths_allowed(url)
  stopifnot(allowed == TRUE)

  local_copy <- read_html(url)
  n_partic <- local_copy %>%
    html_nodes(css = ".header-hero__stat--participants") %>%
    html_text() %>%
    readr::parse_number()

  course_title <- stringr::str_remove(course_link, "/courses/")
    df <- data.frame(date = Sys.Date(),
                     course = course_title,
                     participants = n_partic,
                     stringsAsFactors = FALSE)
  Sys.sleep(1)
  return(df)
}
```

Solution: exercise 2

Get the number of participants for 5 courses

```
purrr::map_df(all_course_links[1:5], get_participants)
```

```
##         date                                   course participants
## 1 2018-09-27                  free-introduction-to-r       962624
## 2 2018-09-27 data-table-data-manipulation-r-tutorial        51647
## 3 2018-09-27      dplyr-data-manipulation-r-tutorial        83098
## 4 2018-09-27     ggvis-data-visualization-r-tutorial        39073
## 5 2018-09-27               reporting-with-r-markdown        52342
```

Webscraping workflow

Prepare

- URL construction
- Allowed to scrape?

`paste0()`
`httr::modify_url()`

`robotstxt::paths_allowed()`

Load info

get a local version

`rvest::read_html()`

Select info

grab the node(s)

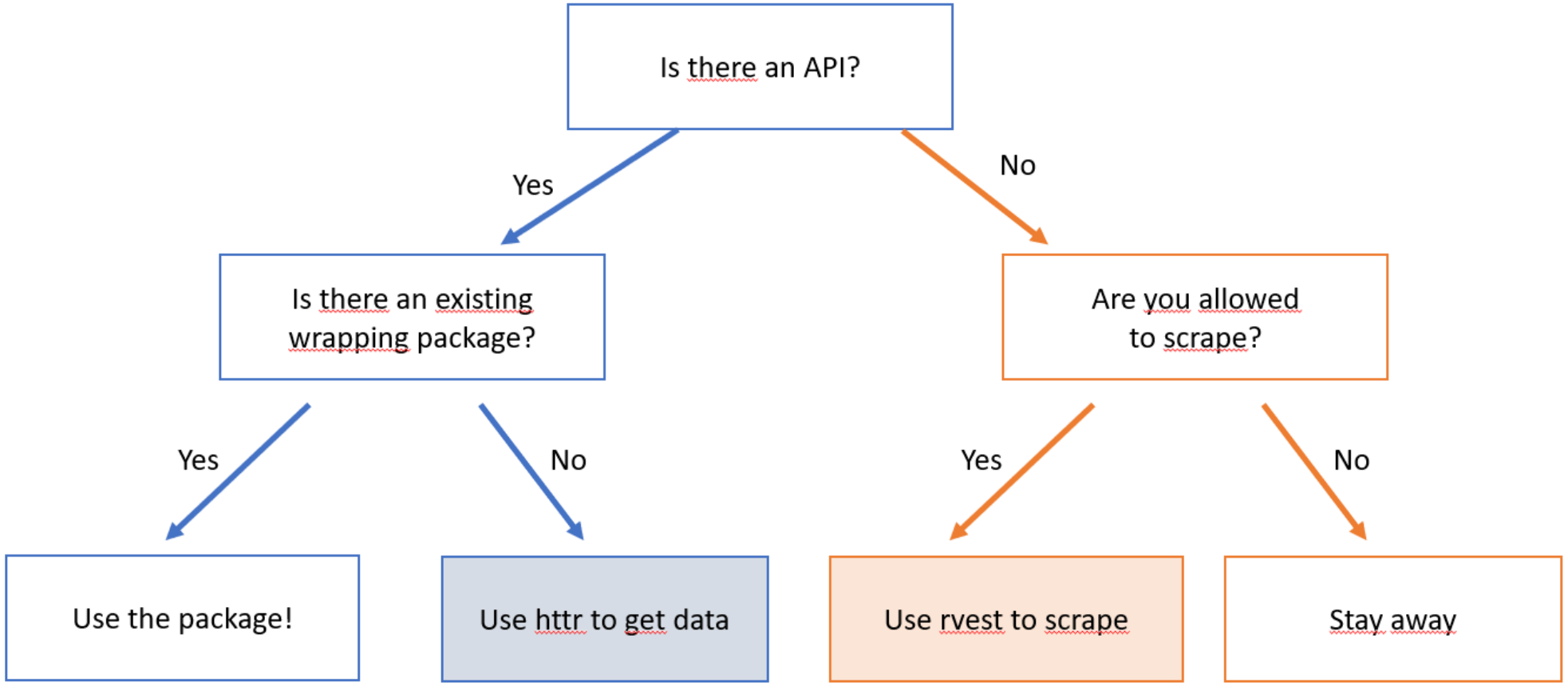
`rvest::html_node()`
`rvest::html_nodes()`

Extract

and parse to R

`rvest::html_text()`
`Rvest::html_attr()`
`rvest::html_table()`

Getting data from the web



Webscraping Etiquette

1. If there is a public API available, use the API and don't scrape the info
2. Always check with `robotstxt` or `polite` whether you are allowed to scrape a page
3. Build system sleeps into your API or scraping requests
4. Don't gather data you won't need
5. Keep away from gathering PII (personal identifiable) data

Other Resources

polite webscraping
package `polite`

Scraping including forms:
https://stat4701.github.io/edav/2015/04/02/rvest_tutorial/

Tools for Javascript rendered websites

- Webshot package
- RSelenium package