Scraping Structured Formats

SICSS

CREST

INSTITUT POLYTECHNIQUE DE PARIS

Before we begin

Introduction

Where are we now?

- We know how to imitate a browser, and consequently to play collect html from a webpage
- We know how to restructure this information into a tree
- But it still looks bulky and we would like to extract relevant information
- (ex. on a website about movies, only retain titles and directors of films)

SICSS CREST INSTITUT POLYTECHNIQUE DE PARIS

Introduction

From the page to the relevant data

Two main options

- Selecting from the structure of the page
- > Using requests that take advantage of the structure of the page

- Selecting from regularities in the text
- > Using regular expressions (tomorrow)

SICSS CREST INSTITUT POLYTECHNIQUE DE PARIS

Introduction

From the page to the relevant data

Two main options

Using Xpath (and css)

- Xpath is a language (yet another one!) to make queries
- Xpath queries the tree structure of a page
- From that, it selects elements into this structure

Introduction - Outline

Principles of Xpath

- A quick reminder on HTML (and markup languages)
- A path to a node
- There are more than one path

Selection

- How to select
- What to select
- How to in R

Remarks on Automation

- A few handy functions
- Inspect your code
- Cheat!

Principles of Xpath

Reminder about markup languages

- HTML is a markup language

- <html>
  - <body>
    - <h1> Hey there </h1>
    - <p> This is a paragraph with a link
    - </body>

</html>

Principles of Xpath

Reminder about markup languages

- HTML is made up of nodes (two tags), which have
  - A type (a, span, div, etc)
  - A content
  - They may have attributes (href,class, id,etc)

> We want to select the content OR the attributes

```
<html>
    <body>
    <h1> Hey there </h1>
    <p> This is a paragraph with <a href="link"> a link </a>
    </body>
    </html>
```

Principles of Xpath

Reminder about markup languages

- An example of XML

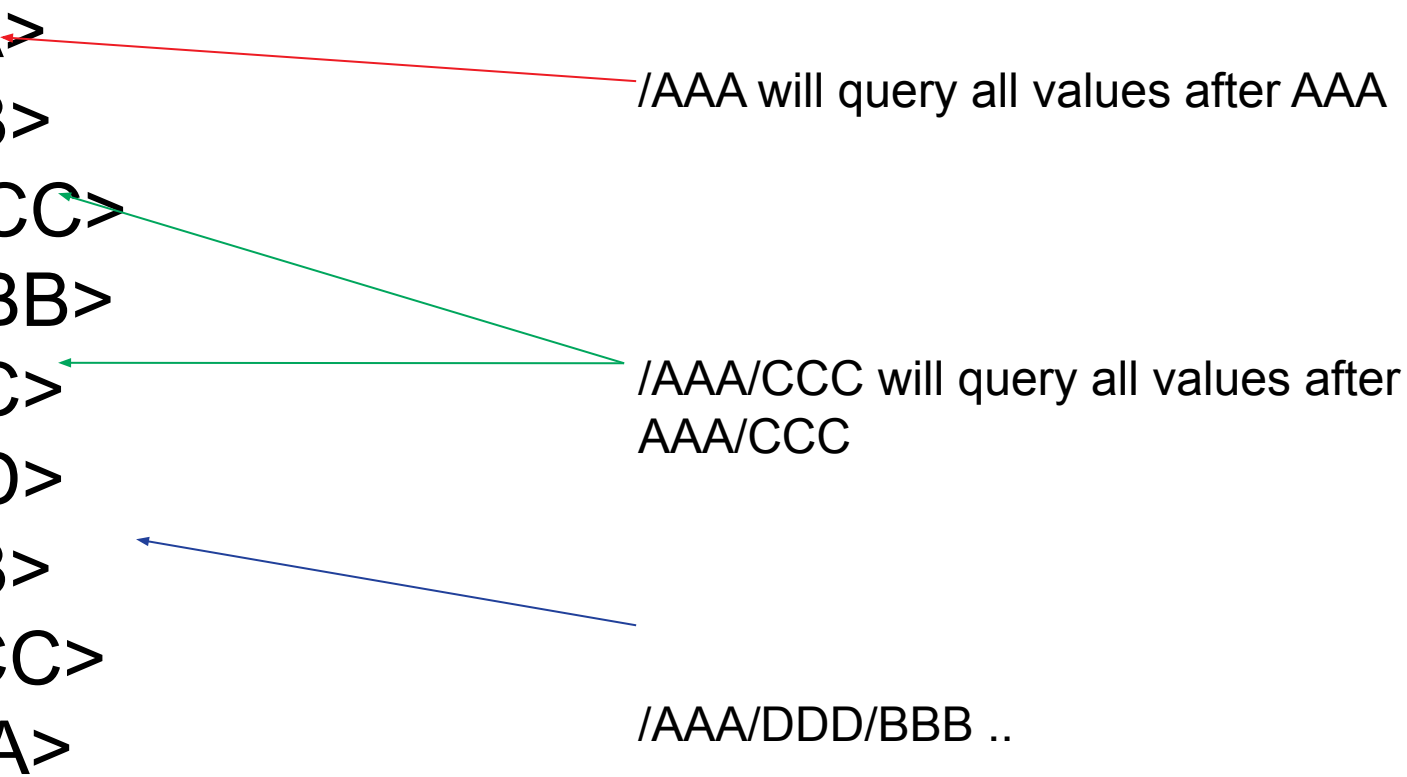
```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>
 <book>
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
 </book>
</bookstore>
```

Principles of Xpath

Xpath, a path to a node

- Xpath is nothing more than a path (= “an address”)

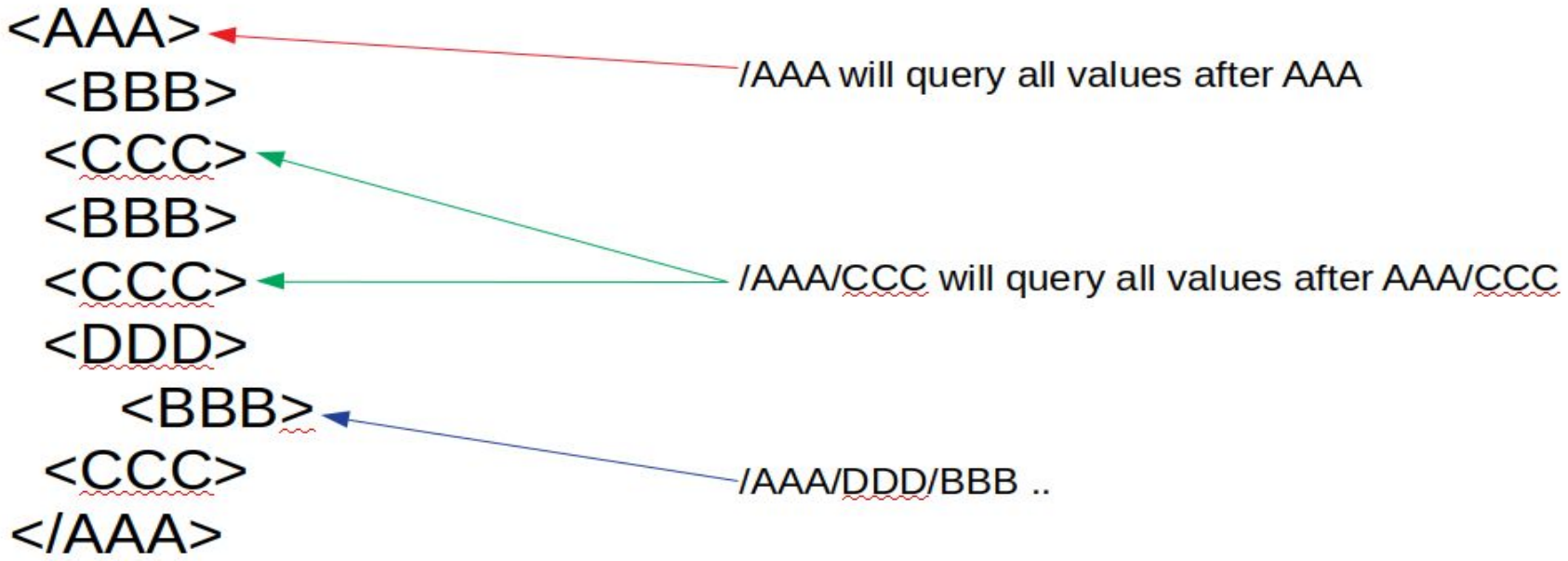
- <AAA>
  <BBB>
    <CCC>
    <BBB>
  <CCC>
  <DDD>
  <BBB>
    <CCC>
  </AAA>



Principles of Xpath

There are several paths to one node

- The simplest path goes from the root to the node
- But you can select from any node (//DDD/BBB)
- You can also skip a node (/AAA/BBB)
- But pay attention: one path may select several nodes



Selection in Xpath

Selection in Xpath

There are three principles of selection

- Absolute (all the way down) / relative
- Going down, going up
- Selecting from content ←99% of cases

Selection in Xpath

There are three principles of selection

- *Absolute vs. relative*

- With the absolute path, you write the path all the way down (your path starts with /)

- With the relative, you start at a point in the tree, and you continue the path (your path starts with //)

Selection in Xpath

There are three principles of selection

- *Going down / Going up*

- While going down may be the most common way, sometimes you have to start at one point, and climb your way up.

Selection in Xpath

There are three principles of selection

- *Selection from content*
- Most of the time, we only want to select the nodes that match a certain conditions (all the titles in a page)

- This condition is written between [ ], and is related to the text, or to an attribute (@class, @href, ...)

  - `//div[@class = “book”]` : content from all *div* whose *class* is *book*

Selection in Xpath

Four main operators

- / Child → Direct descendant
- // Descendant → Anywhere below
- [ ] Condition
- @ Attribute

How to select

What to select

Selection in Xpath

Xpath	Action
/a	
//a	
//div/a	
//div//a	
//a[contains(text(), ‘sociology’)]	All the links that contain ‘sociology’

Selection in Xpath

Xpath	Action
/a	All hypertext links **at the root level**
//a	All the hypertext links
//div/a	All the hyperlinks in the div (just after)
//div//a	All the hyperlinks anywhere in the div
//a[contains(text(), ‘sociology’)]	All the links that contain ‘sociology’

Selection in Xpath

What to select

- **Distinction between value and attribute**
  - **Value**: what is between the two tags
  - <TAG>content</TAG>

  - **Attribute**: what is contained inside the tag
  - Blablabla

Selection in Xpath

To go further

- Joker: /* → any type of tag
- Ex. /div/*/a

- Conditions: contains(), starts-with(), ends-with()

- Navigation: child(), parent(), following-sibling()

- Number of descendants: count()

How-to in R

A few handy functions

- **html_elements(PAGE, xpath='PATH')**

  PAGE: the (restructured) page
  PATH: the path

How-to in R

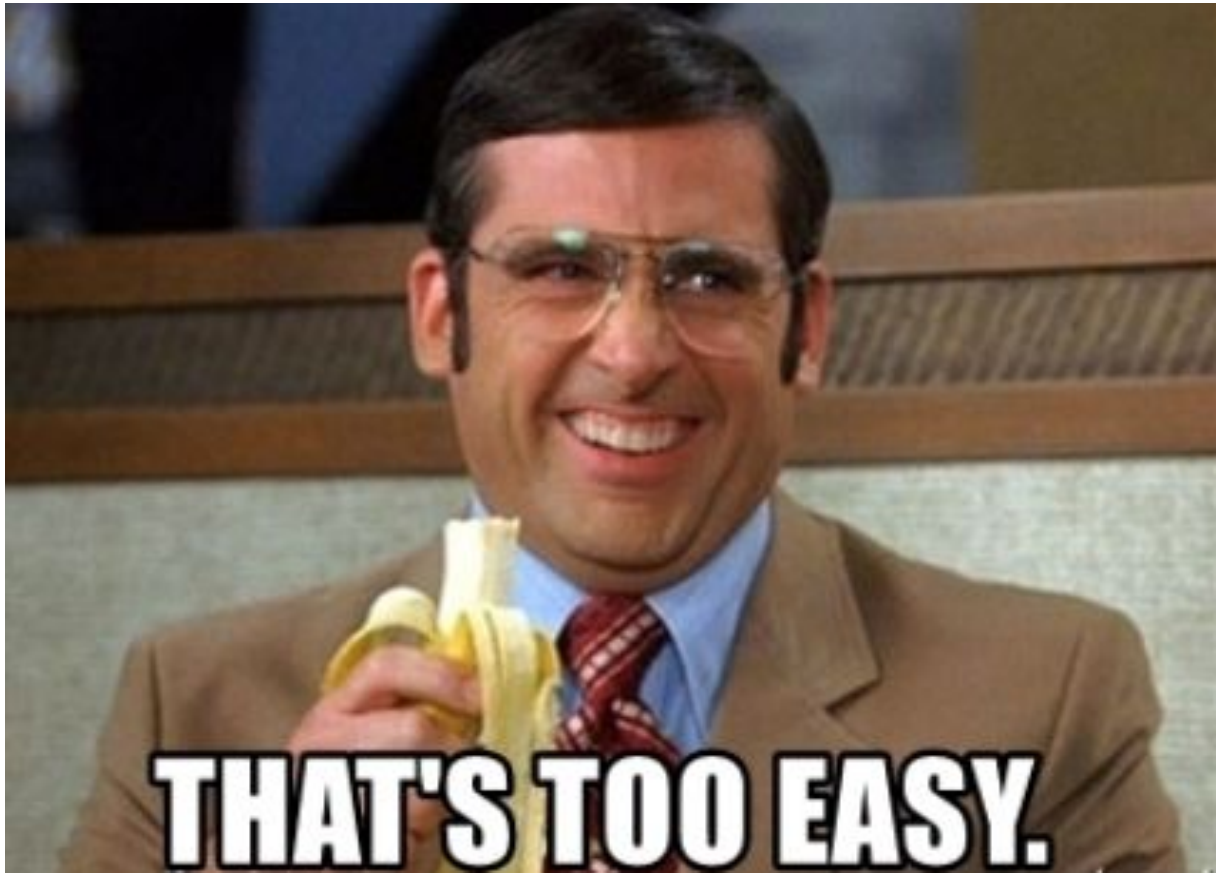
A few handy functions

- **So:**

- read_html("https://sicss.io/2022/paris/schedule")
  %>%
  html_elements(xpath='//li')

  will return all elements stored as a list

How-to in R



How-to in R

A few handy functions

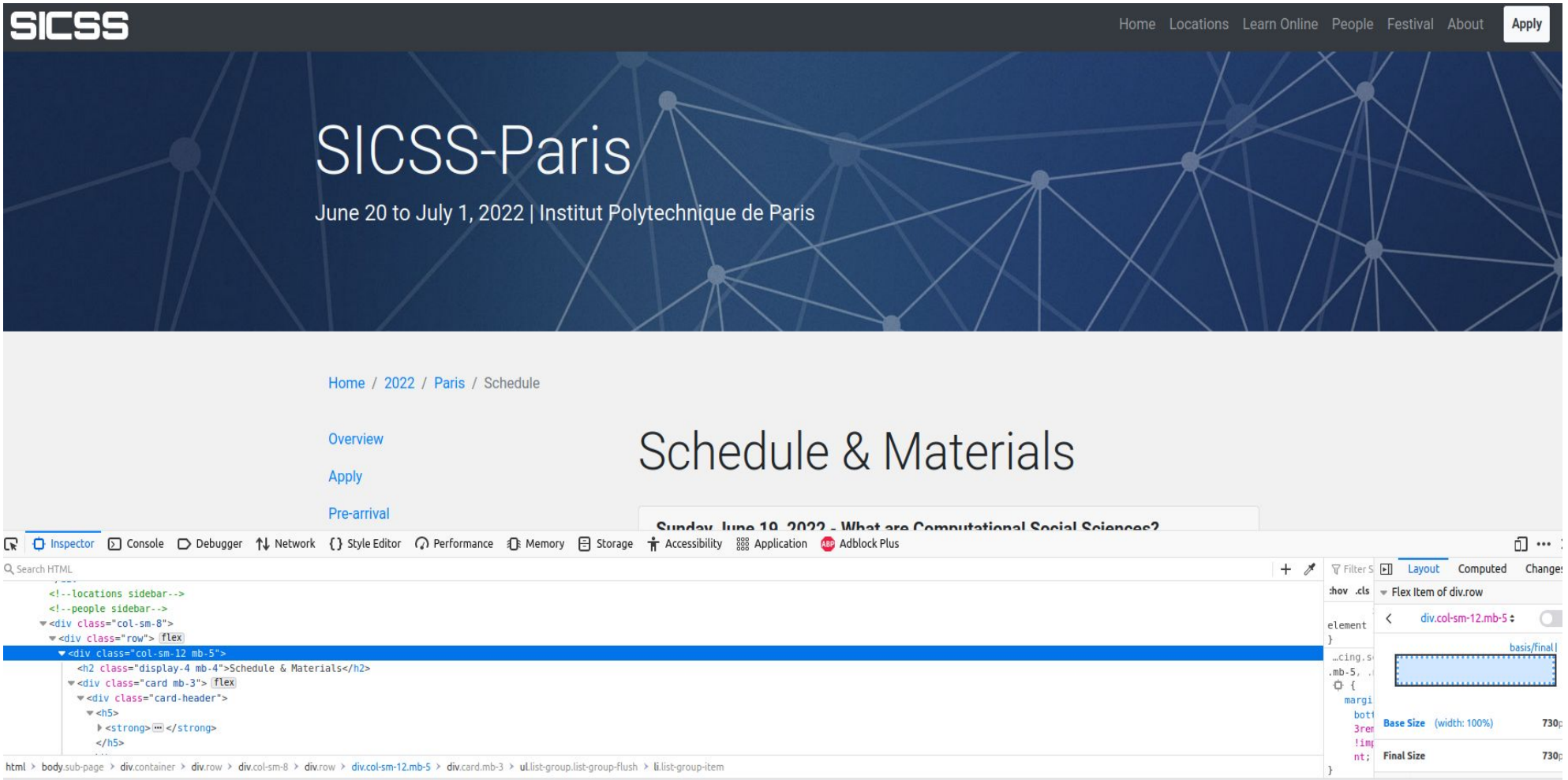
- html_table() #Extracts tables
- html_text() #Extracts raw text (faster)
- html_text2() #Uses formating, nicer
- html_forms() #need to add a login, enter a query?

And one important extra function

- html_attrs("href") #To get the attributes

How-to in R

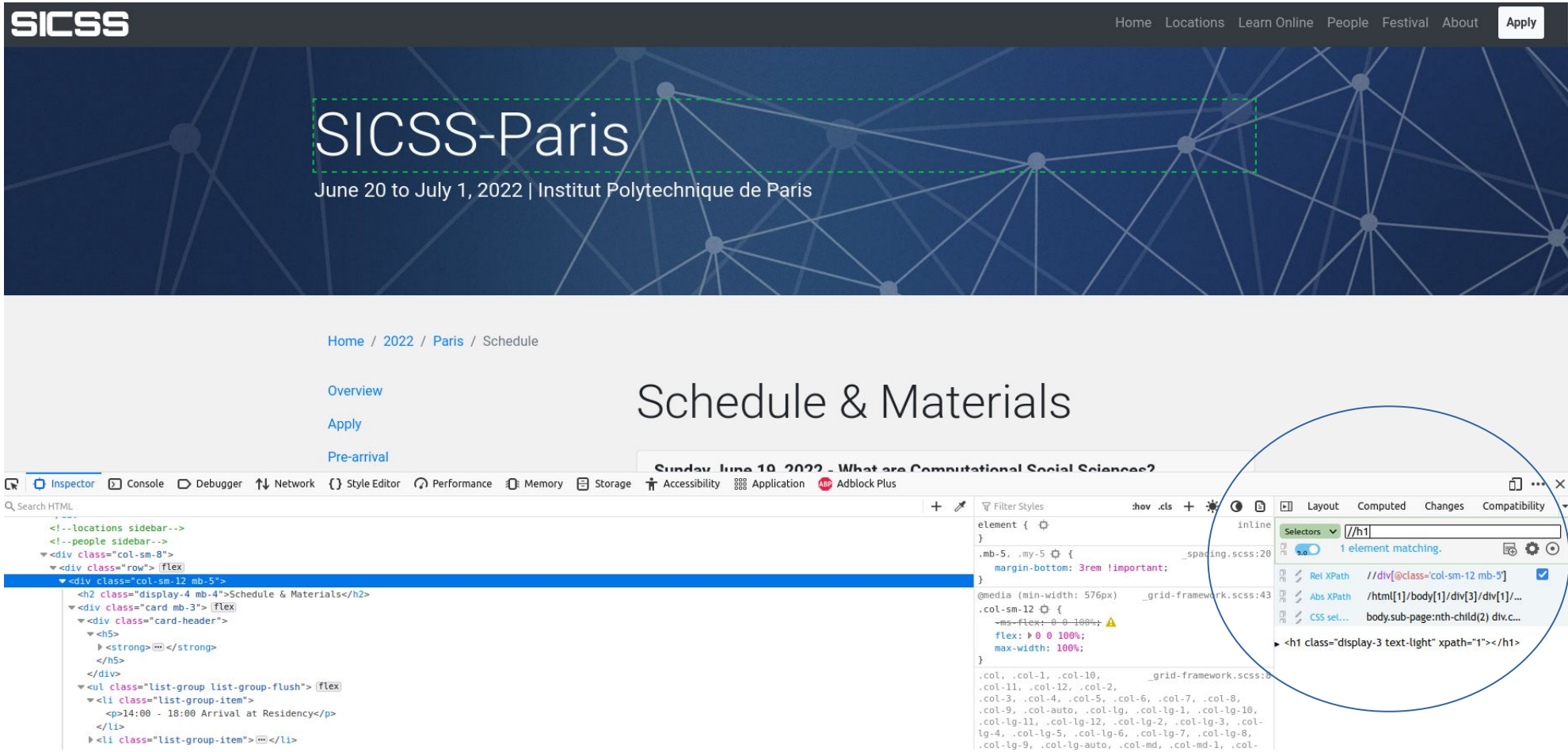
Inspect your code



How-to in R

Install (and use) ChroPath

Useful to validate your xpath queries



How-to in R

Often, it’s even easier to use css selectors

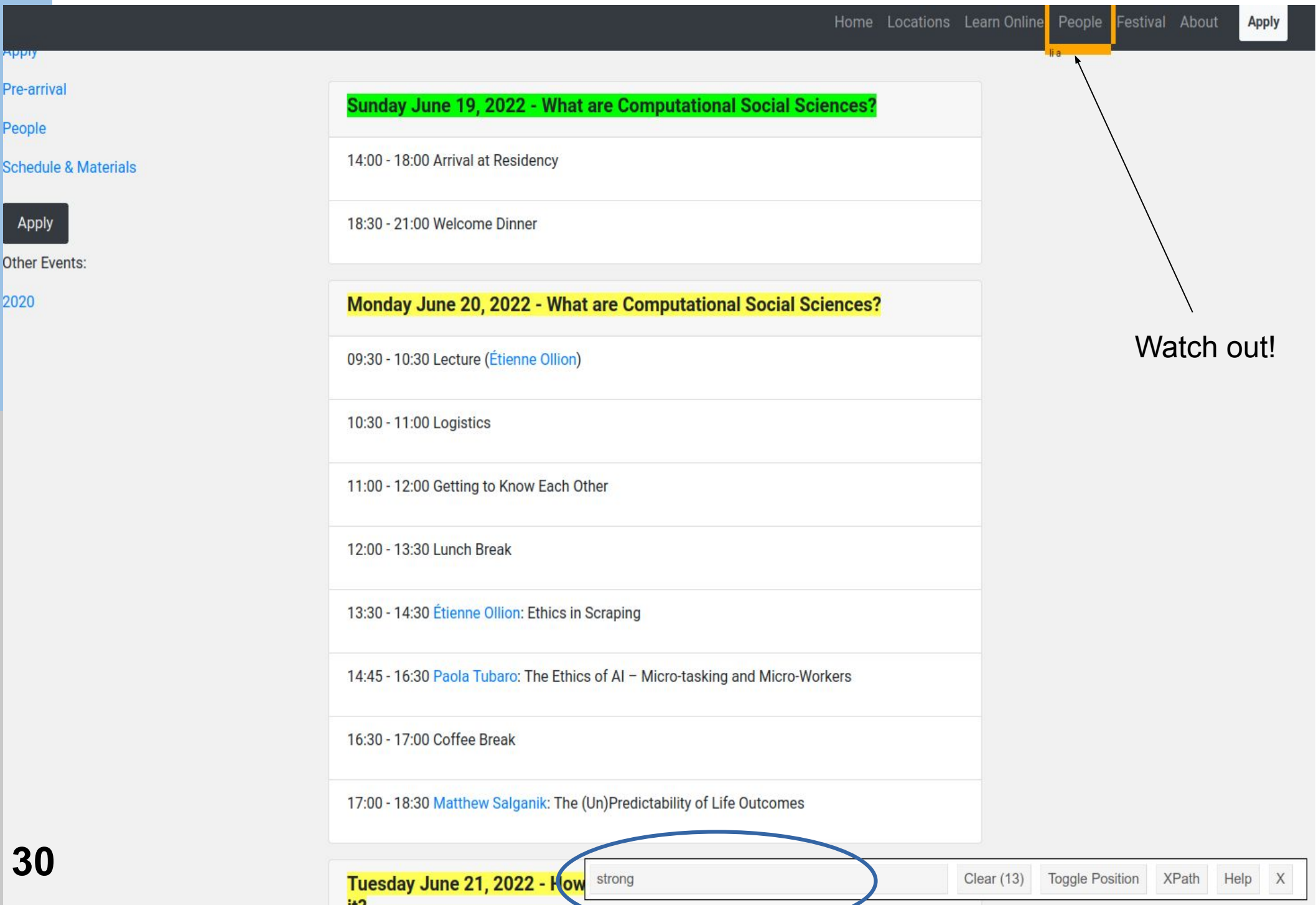
Like xpath, css selectors select html elements according to their type.

How-to in R

Often, it's even easier to use css selectors

Like xpath, css selectors select html elements according to their type.

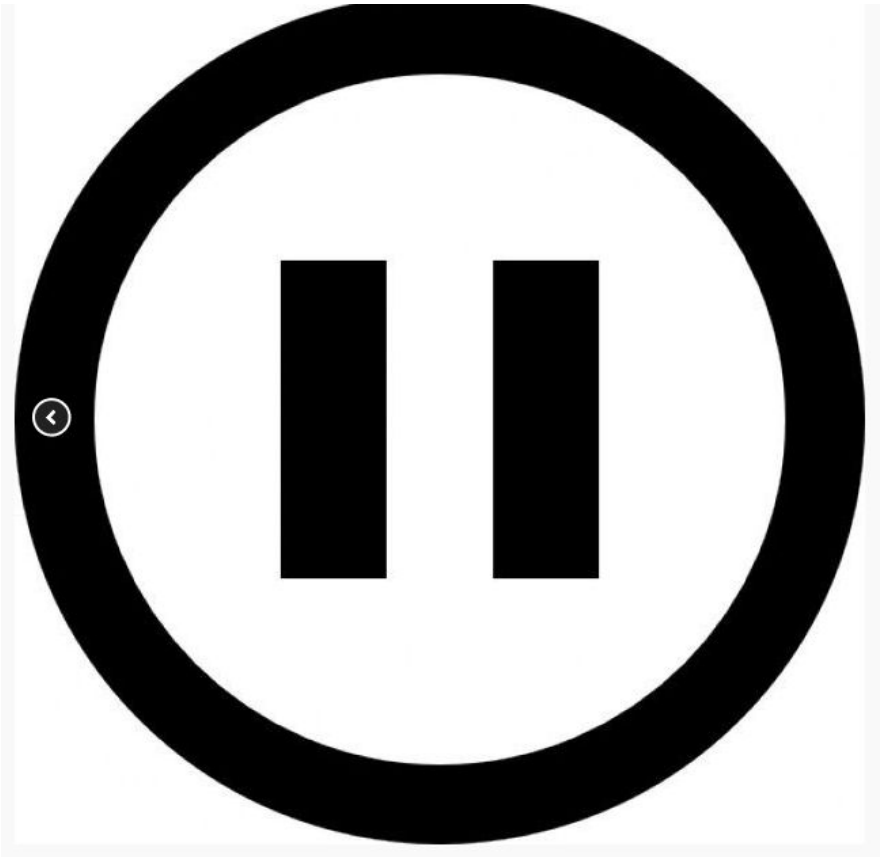
And you can cheat using Selector Gadget


Home Locations Learn Online People Festival About Apply
li a
Apply
Pre-arrival
People
Schedule & Materials
Apply
Other Events:
2020
Sunday June 19, 2022 - What are Computational Social Sciences?
14:00 - 18:00 Arrival at Residency
18:30 - 21:00 Welcome Dinner
Monday June 20, 2022 - What are Computational Social Sciences?
09:30 - 10:30 Lecture (Étienne Ollion)
10:30 - 11:00 Logistics
11:00 - 12:00 Getting to Know Each Other
12:00 - 13:30 Lunch Break
13:30 - 14:30 Étienne Ollion: Ethics in Scraping
14:45 - 16:30 Paola Tubaro: The Ethics of AI – Micro-tasking and Micro-Workers
16:30 - 17:00 Coffee Break
17:00 - 18:30 Matthew Salganik: The (Un)Predictability of Life Outcomes
Tuesday June 21, 2022 - How
strong
Clear (13)
Toggle Position
XPath
Help
X
Watch out!

How-to in R

- read_html("https://sicss.io/2022/paris/schedule") %>%
  html_elements(css="strong")

```
> read_html("https://sicss.io/2022/paris/schedule")%>%
+   html_elements(css="strong")
{xml_nodeset (13)}
 [1] <strong>\n\t  \n          Sunday June 19, 2022\n\t  \n          \n           - What are Computational Social Sciences?\n       ...
 [2] <strong>\n\t  \n          Monday June 20, 2022\n\t  \n          \n           - What are Computational Social Sciences?\n       ...
 [3] <strong>\n\t  \n          Tuesday June 21, 2022\n\t  \n          \n           - How is the Web Written and How to Collect Data ...
 [4] <strong>\n\t  \n          Wednesday June 22, 2022\n\t  \n          \n           - Scraping Structured Formats\n          \n    ...
 [5] <strong>\n\t  \n          Thursday June 23, 2022\n\t  \n          \n           - Scraping Unstructured Formats\n          \n   ...
 [6] <strong>\n\t  \n          Friday June 24, 2022\n\t  \n          \n           - Natural Language Processing\n          \n       ...
 [7] <strong>\n\t  \n          Saturday June 25, 2022\n\t  \n          \n           - off-day\n          \n        </strong>
 [8] <strong>\n\t  \n          Sunday June 26, 2022\n\t  \n          \n           - off-day\n          \n        </strong>
 [9] <strong>\n\t  \n          Monday June 27, 2022\n\t  \n          \n           - Natural Language Processing\n          \n       ...
[10] <strong>\n\t  \n          Tuesday June 28, 2022\n\t  \n          \n           - Natural Language Processing\n          \n      ...
[11] <strong>\n\t  \n          Wednesday June 29, 2022\n\t  \n          \n           - Agent-Based Modeling\n          \n        </ ...
[12] <strong>\n\t  \n          Thursday June 30, 2022\n\t  \n          \n           - Social Experiments in a Digital World\n       ...
[13] <strong>\n\t  \n          Friday July 1, 2022\n\t  \n          \n           - Final day\n          \n        </strong>
```



Automation

Automation

So far, we have seen

- How to replicate a crawler and get the content of an HTTP request
  > read_html()

- How to parse a mark-up language to extract content
  > html_elements() and kin functions

→ But...we have only done this on one page

Automation

We need to

- Learn how to move from one page to another
- Store results
- And we'll see a few tricks about optimization

Automation

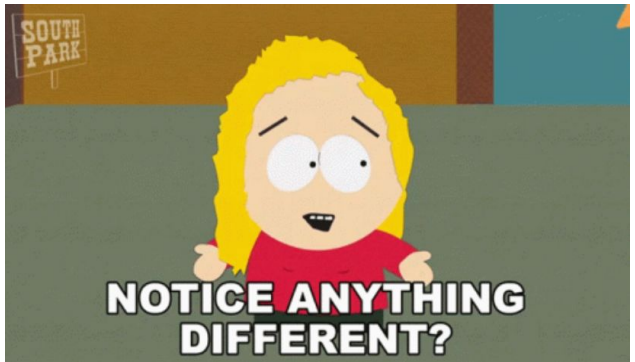
Method 1: via URL

In many cases, and certainly in the oldest versions of the web, the URL reflects what we are asking to a page.

This is very useful when a website is organized in such a way that it will ask you to “turn a page”

On a website that lists plane crashes

- https://aviation-safety.net/database/dblist.php?Year=1919
- https://aviation-safety.net/database/dblist.php?Year=1920
- https://aviation-safety.net/database/dblist.php?Year=1921



Automation

Method 1: via URL

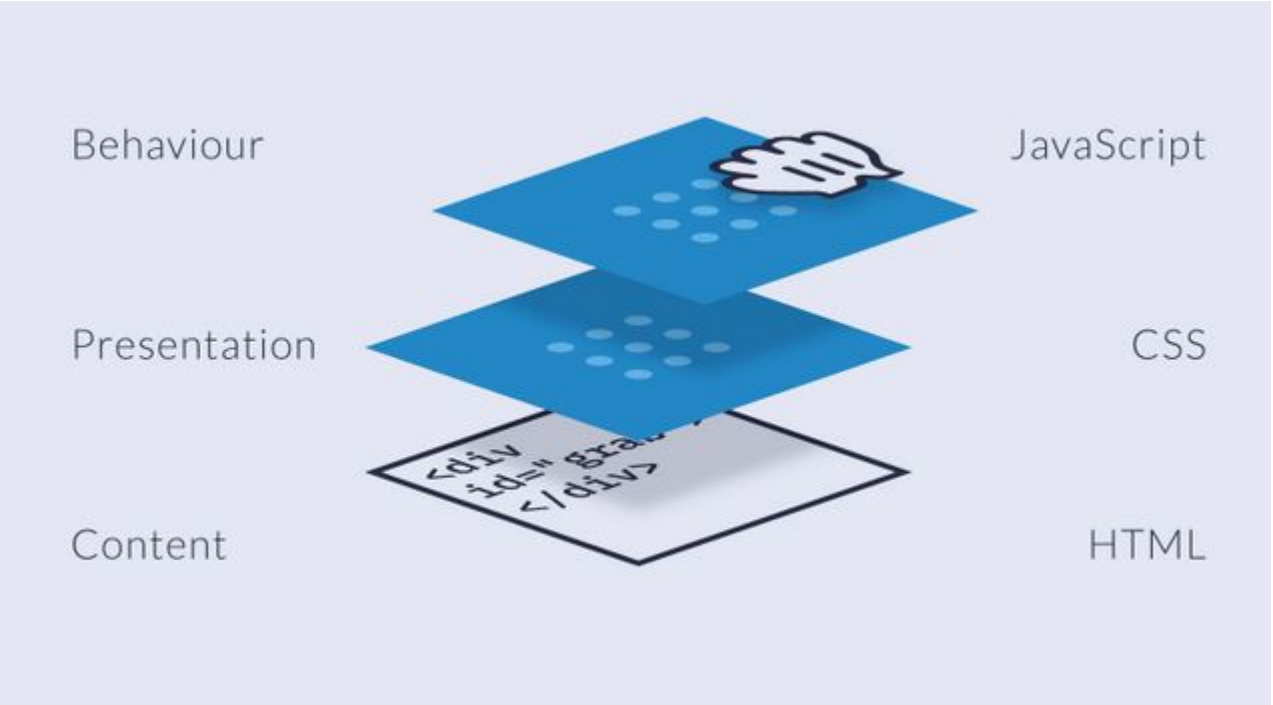
In this case, we just have to create a list of all the URLs we need to visit.

And then, we'll ask our crawler to visit them one by one.

Automation

Method 2: Behavior-based websites

A growing trend in the web industry is to have websites that respond to your behavior (scrolling, clicking, etc).



Automation

Method 2: Behavior-based websites

→ Headless browsers

Automation

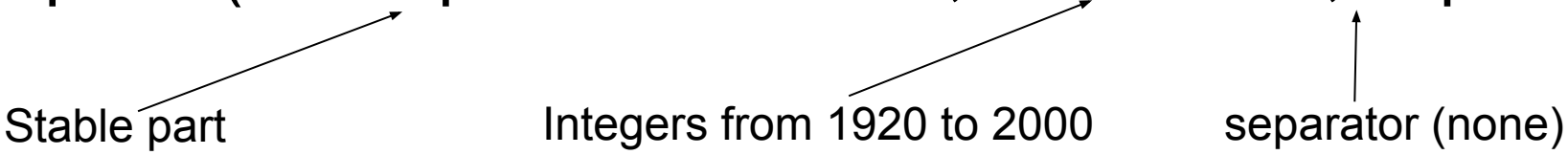
How-to in R

If you go “server-side”, create a list of URLs to visit
Since we have noticed a pattern, we just need to repeat it.

We can use the **paste()** function

paste(STABLE PART, MOVING PART, sep)

paste(“www.planecrashinfo/”, 1920:2000, sep=“”)



Automation

How-to in R

We can use the **paste()** function

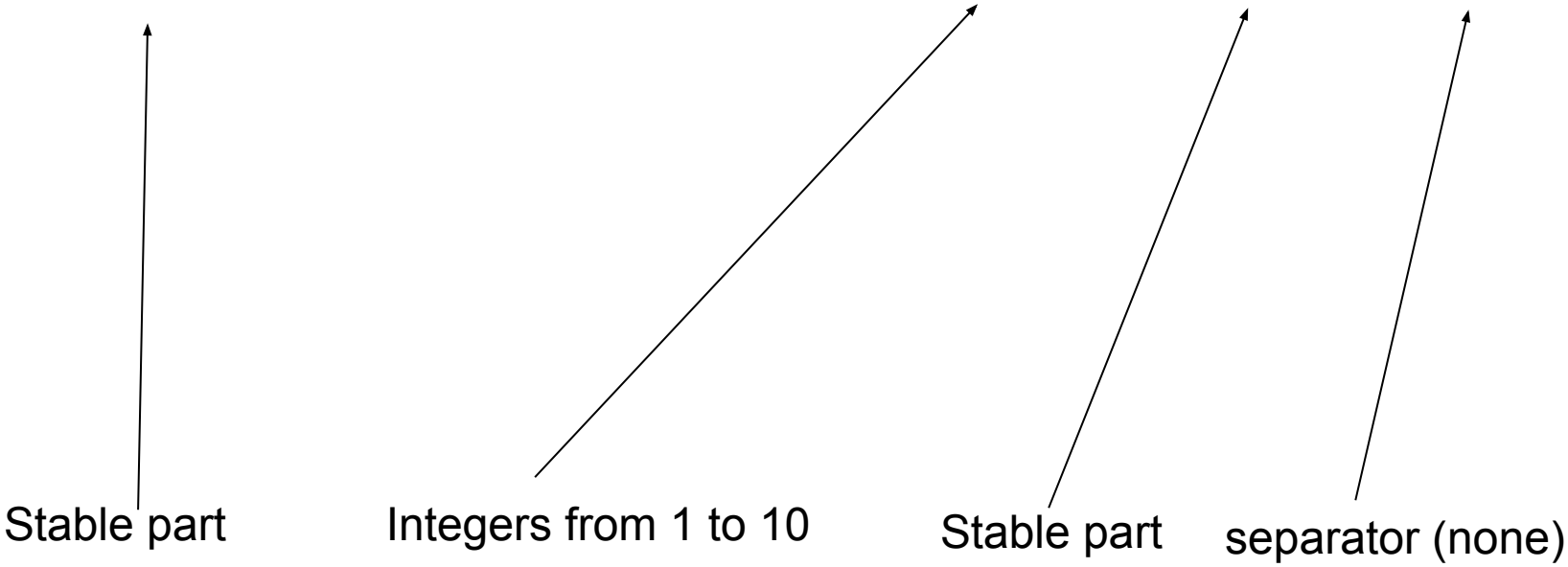
paste(STABLE PART, MOVING PART, STABLE PART, sep)

Automation

How-to in R

You can have a series of fixed parts

paste("http://sf.craigslist.org/index/", 1:10, "00.html", sep="")



Automation

A note on Storage

Automation

Reminder: Formats in R

A **vector** is a list of elements of the same type

```
a ← c(“Bob”, “Alice”, “Carol”)
b ← 1:10
urls <- paste0(“http://www.thepage.org/&pagenb=“, 1:10)
```

Automation

A **matrix** is a table of elements of the same type

```
x ← matrix(1:10, ncol=2)
y ← cbind(c(“a”, “b”, “c”), c(“A”,“B”,“C”))
```

Automation

A **dataframe** is a matrix whose elements can be different

```
my.df ← data.frame(nom=c(“Alice”, “Bob”, “Carole”),
                   id=1:3,
                   score=c(10,5,1))
```

You can pass variable names to a data frame
Note that the vectors have the same size

Automation

A **list** assembles formats of various size and format

```
my_list ← list(nom=c(“Alice”, “Bob”, “Carole”),
                id= “students”,
                score=c(10,5,1,4,9,7))
```

A list is a more flexible format for storing

Automation

‘for’ Loops

An easy (because intuitive way) of storing is to use for loops

For loops repeat an action for all the values of a given vector

```
for (i in 1:5) {print(i)}
```

Automation

‘for’ Loops

But wait, you are not storing anything!

Let’s create a container first:
basket ← NULL
basket[i] ← for (i in 1:5) {print(i)}

Automation

‘for’ Loops

```
urls←paste0(“https://aviation-safety.net/database/dblist.php?Year=”, 1919:2020)

pages ← list() #Creates an empty object

for (i in 1:101) {
    pages[[i]] ← read_html(urls[i]) # Collects content of urls[i], stores it into pages[[i]]
    print(i)                        # Prints the page we are at
    Sys.sleep(1)                    # Pause 1 second
    }
```

Automation

Once at this stage, we know how to extract data from numerous web pages.

But if the computer works for you, it also has limited capacities, and you can try to make it act in an efficient way.

This is what algorithmics is about. This is a field in itself.
For us:

- Avoid (double) loops if you have thousands of operations
- Pre-define your storage
- Can you parallelize your tasks?
- Do you need more computer power?

Cf. an old but good book:Burns, R Inferno (2011)

Conclusion

Ok, that was a lot.

But in the end:

1. Create a vector of webpages you want to visit
2. Collect them, extract the content
3. Store this

This can be a matter of 4 lines of code.

We will practice this afternoon