Workshop Web-Scraping in R

Simon Ress

Ruhr-Universität Bochum

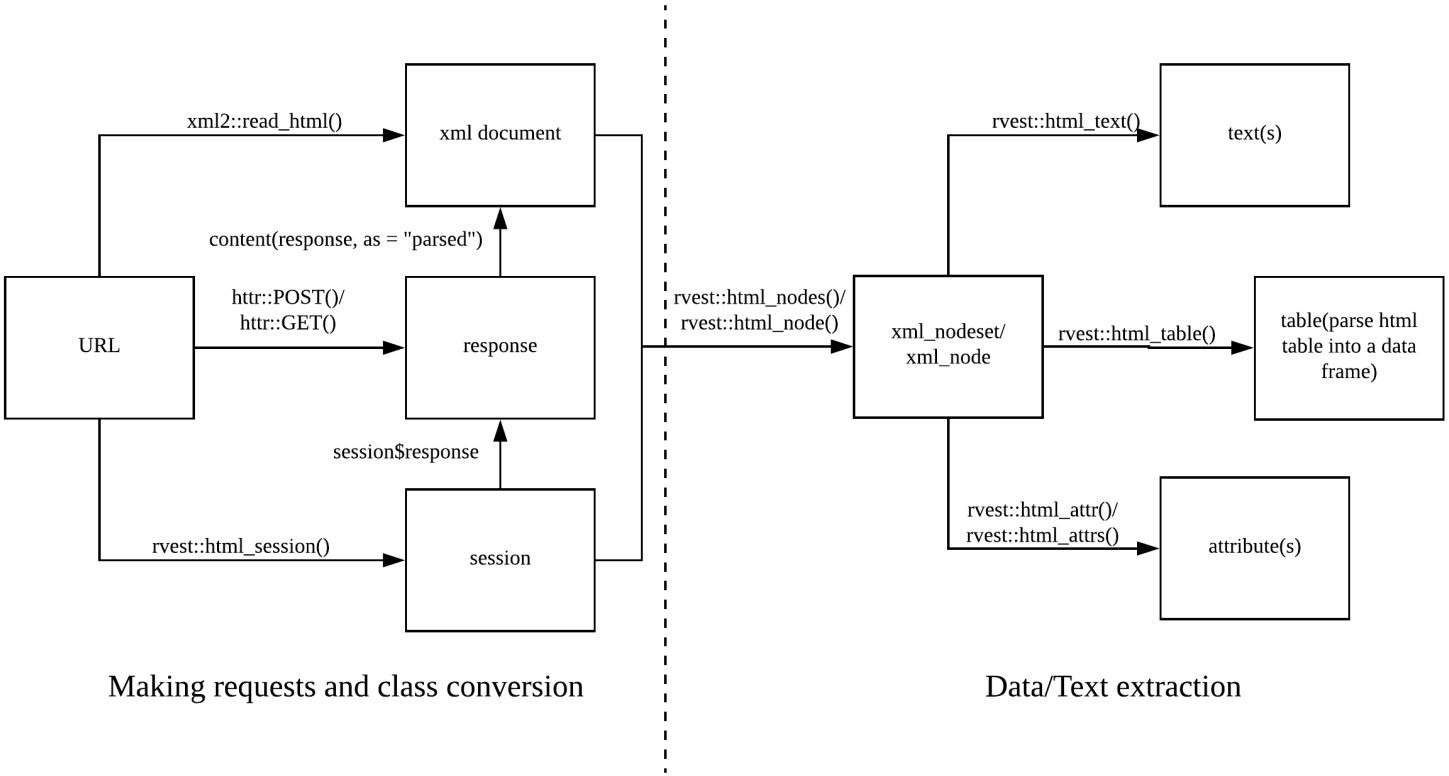
15.02.2020

Inhalt

Chap. 1 Übersicht

Übersicht Web-Scraping

Figure: Schema von Web-Scraping & zentrale Befehle



Quelle: https://github.com/yusuzech/r-web-scraping-cheat-sheet

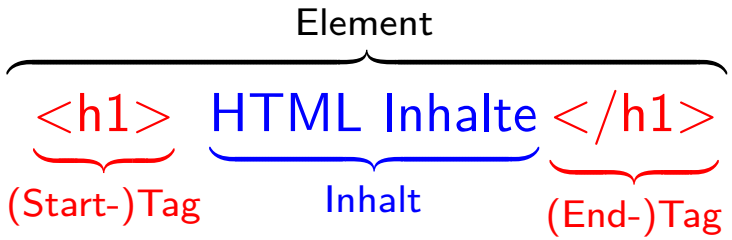
Chap. 2 HTML

HTML: Was ist das?

- Hypertext Markup Language (HTML) ist eine textbasierte Auszeichnungssprache
- Dient der Strukturierung von Webseiten
- Besteht aus einer Reihe von Elementen (→ definieren Struktur)
- Elemente regeln die Darstellung durch den Browser
- Elemente bestimmen; "das ist eine Überschrift", "das ist ein Paragraph", "das ist ein Link" usw.

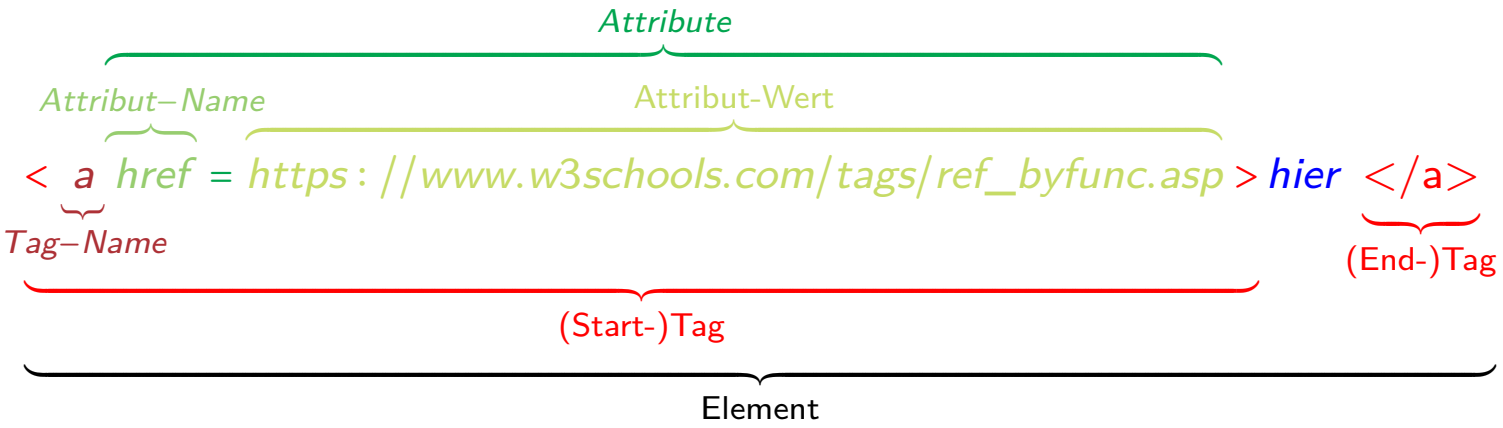
HTML: Elemente, Tags und Attribute I

- **Tags:** Start- (<Tag-Name>) und End-Punkte (</Tag-Name>) von Elementen.
- **Element:** Zusammenspiel aus darzustellendem Inhalt und Darstellungsvorgabe für Browser



HTML: Elemente, Tags und Attribute II

- **Attribute:** Optionen der Elemente, welche in Start-Tags definiert werden (bestehen aus Attribut-Name & -Wert)

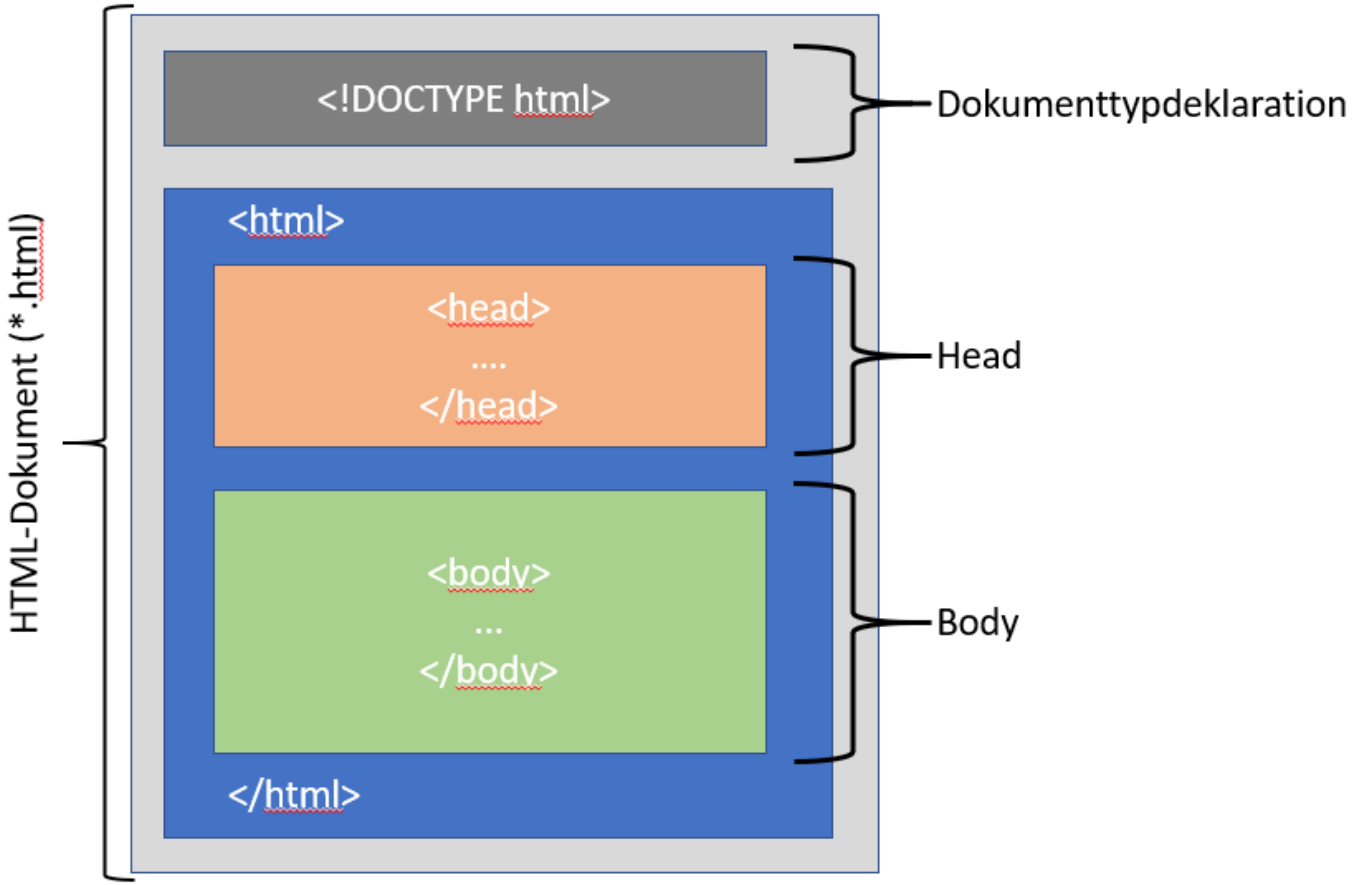


HTML: Grundstruktur

Ein HTML-Dokument besteht aus drei Bereichen:

- **Dokumenttypdeklaration (<!DOCTYPE html>):** Beginn der Datei, die die verwendete Dokumenttypdefinition (DTD) angibt, z. B. HTML oder CSS
- **HTML-Kopf (<head>):** Enthält hauptsächlich technische oder dokumentarische Informationen, die üblicherweise nicht im Anzeigebereich des Browsers dargestellt werden
- **HTML-Körper (<body>):** Enthält alle Informationen die gewöhnlich im Anzeigebereich des Browsers zu sehen sind

HTML: Grafik der Grundstruktur



Eigene Darstellung

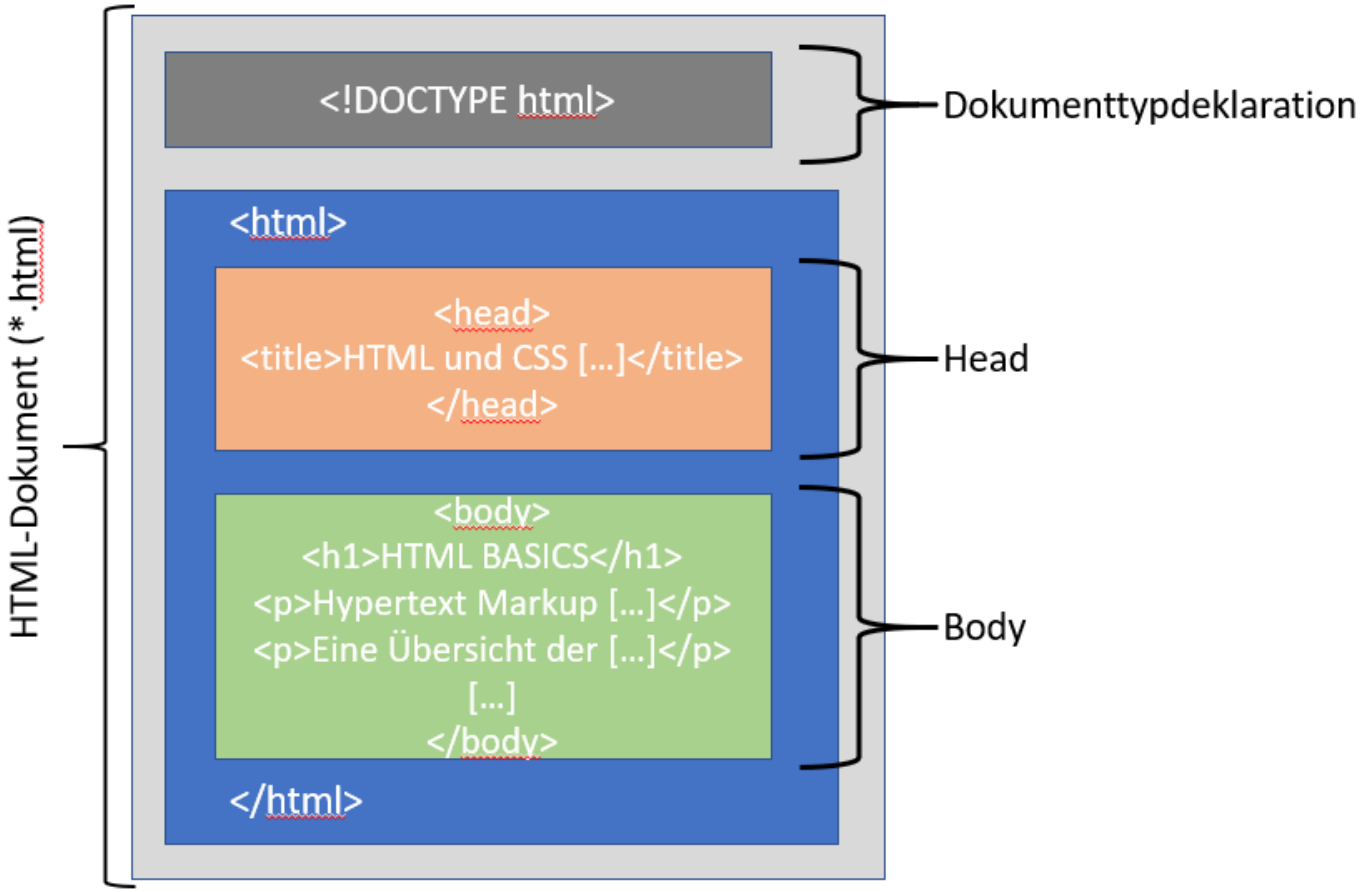
Quelle:

HTML: Überblick von Elementen

Element-Name	Beschreibung
<title>	Titel des Dokuments
<h1> bis <h6>	Überschriften in absteigender Ebene
<p>	Paragraph
<b>, <i>, <u>	Text fett, kursiv, unterstrichten
<a>	Hyperlink
/	Ungeordnete / geordnete Liste
	Item einer Liste
<picture>	Bild einfügen
<video>	Video einfügen (Medien Element)
<audio>	Sound einfügen (Medien Element)
<source>	Datei für Medien Element

Hier eine ausführliche Liste von HTML-Elementen nach Verwendungszweck.

HTML: Basics.html (Schema)



Quelle:

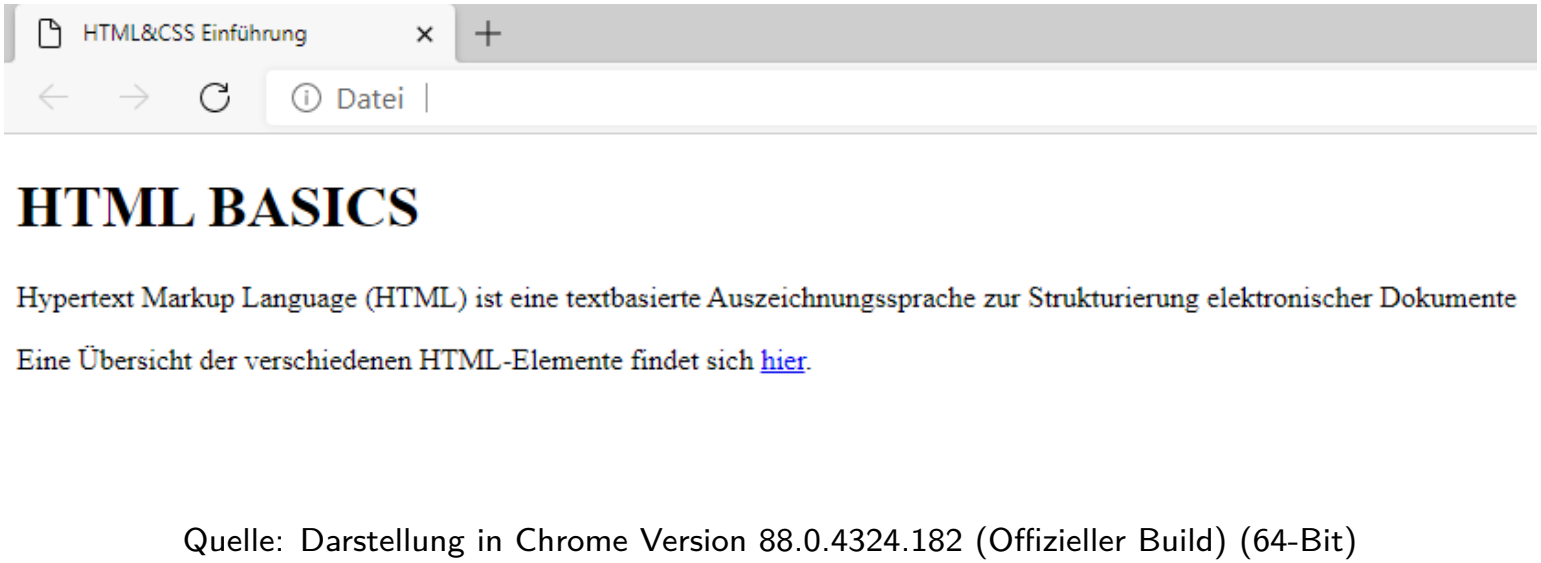
Eigene Darstellung

HTML: Basics.html (Code)

```
<!DOCTYPE html>
  <html>
    <head>
      <title>HTML und CSS Einführung</title>
    </head>

    <body>
      <h1>HTML BASICS</h1>
      <p>Hypertext Markup Language (HTML) ist eine
         textbasierte Auszeichnungssprache zur
         Strukturierung elektronischer Dokumente</p>
      <p>Eine Übersicht der verschiedenen HTML-
         Elemente findet sich <a href="
         https://www.w3schools.com/tags/ref_byfunc.asp"
         >hier</a>.</p>
      [...]
    </body>
  </html>
```

HTML: Basics.html (Browser)

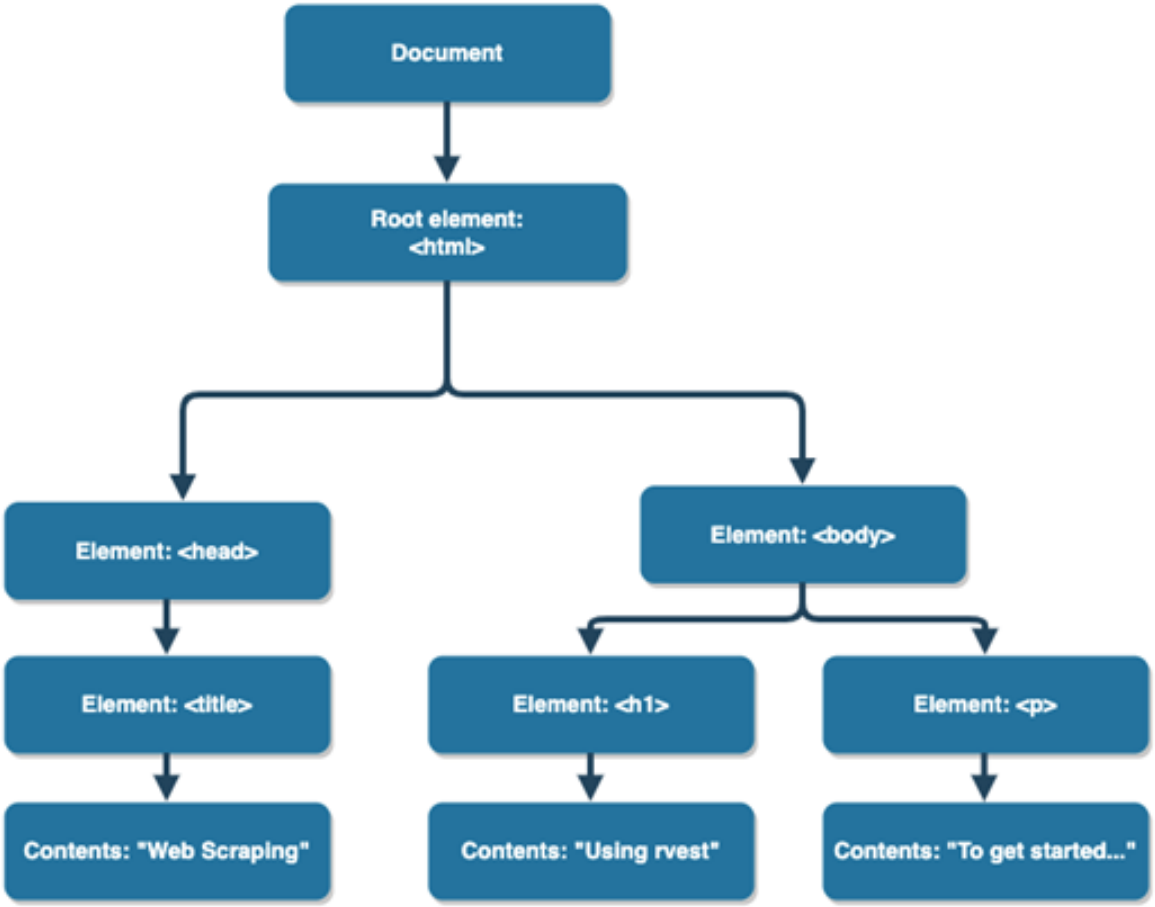


Quelle: Darstellung in Chrome Version 88.0.4324.182 (Offizieller Build) (64-Bit)

Link:

HTML: Baumstruktur

Die Verschachtelung der Inhalte eines HTML-Dokuments kann am besten mittels eines Baumdiagramms dargestellt werden.



Quelle:

Chap. 3 CSS

CSS-Grundlagen

- Cascading Style Sheets (CSS) ist eine Stylesheet-Sprache für elektronische Dokumente und zusammen mit HTML und JavaScript eine der Kernsprachen des World Wide Webs.
- CSS wurde entworfen, um Darstellungsvorgaben weitgehend von den Inhalten zu trennen.
- Layouts, Farben und Typografie von Websites können mit Cascade Style Sheets formatiert werden (Trennung von Inhalte=HTML/XML und Layout=CSS)
- Speicherung in separaten CSS-Dateien
- Darstellung von immer wiederkehrenden HTML-Inhalten erleichtert werden, indem Stile, Farbe und Formen für Titel, Schriftgröße usw. definiert werden können
- Aufbau nach id, class & attribute

CSS-Struktur

Aufbau einer CSS-Anweisung:

```
Selektor1 [, Selektor2 [, …] ] {
    Eigenschaft-1: Wert-1;
    …
    Eigenschaft-n: Wert-n[;]
}
/* Kommentar */
/* In eckigen Klammern stehen optionale Angaben */
```

CSS-Struktur: Beispiel

Beispiel einer CSS-Anweisung:

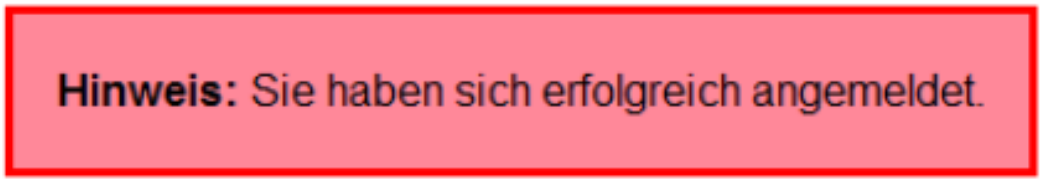
```
p.info { #ID:p // class: info
  font-family: arial, sans-serif;
  line-height: 150%;
  margin-left: 2em;
  padding: 1em;
  border: 3px solid red;
  background-color: #f89;
  display: inline-block;
}
p.info span {
  font-weight: bold;
}
p.info span::after {
  content: ": ";
}
```

Einbindung von CSS in HTML

HTML-Code:

```
<p class="info">
  <span>Hinweis</span>
  Sie haben sich erfolgreich angemeldet.
</p>
```

Ergebnis im Browser:

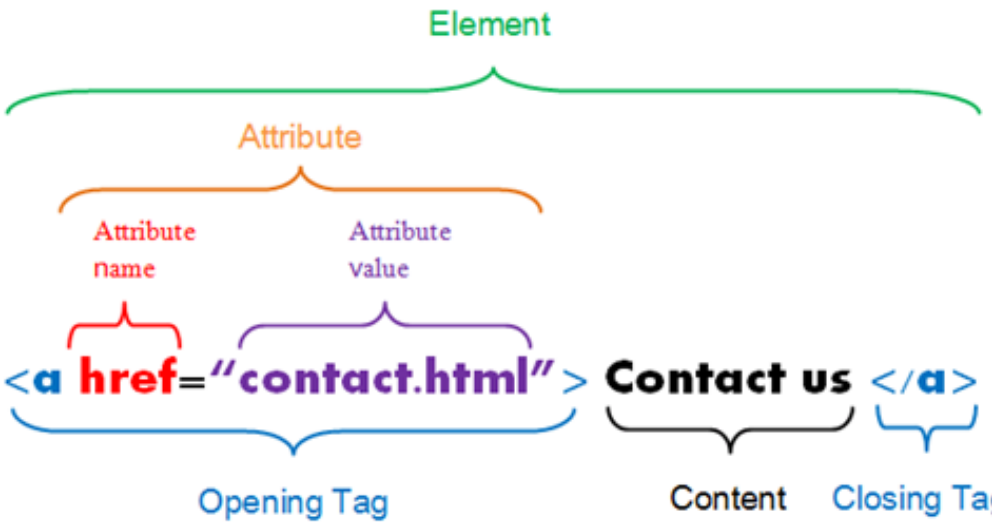


Quelle: Eigene Darstellung

CSS: id, class & attribute

HTML-Code:

```
<div class="hallo-beispiel">
    <a href="contact.html">Contact us:</a>
    <span lang="en-us en-gb en-au en-nz">
      Hallo Welt!
    </span>
</div>
```



Quelle: Eigene Darstellung

CSS-Selector

In CSS sind selectors Muster, mit denen die Elemente ausgewählt werden, die abgefragt werden sollen.
Hier ein erster Einblick: CSS Selector Tester

CSS-Selector: Übersicht I

Selector	Beispiel	Beschreibung
.class	.intro	Alle Elemente mit class="intro"
.class1.class2	.name1.name2	Alle Elemente mit "name1" & "name2" in class attribute
		Alle Elemente mit class="name2"
.class1 .class2	.name1 .name2	welche Elementen mit class="name1" untergeordnet sind
#id	#firstname	Alle Elemente mit id="firstname"
*	*	Alle Elemente
element	p	Alle <p>Elemente
element.class	p.intro	Alle <p>Elemente mit class="intro"

CSS-Selector: Übersicht I

Selector	Beispiel	Beschreibung
element>element	div >p	Alle Elemente <p>deren parent Elemente sind
element>element	div p	Alle <p>-Elemente in -Elementen
[attribute]	[target]	Alle Elemente mit target attribute
[attribute=value]	[target=_blank]	Alle Elemente mit target=_blank
[attribute^=value]	a[href^="https"]	Alle Elemente deren href attribute mit "https" beginnt
[attribute$=value]	a[href$=".pdf"]	Alle Elemente deren href attribute mit ".pdf" endet
[attribute*=value]	a[href*="w3"]	Alle Elemente deren href attribute "w3" enthält

CSS-Selector & XPaths

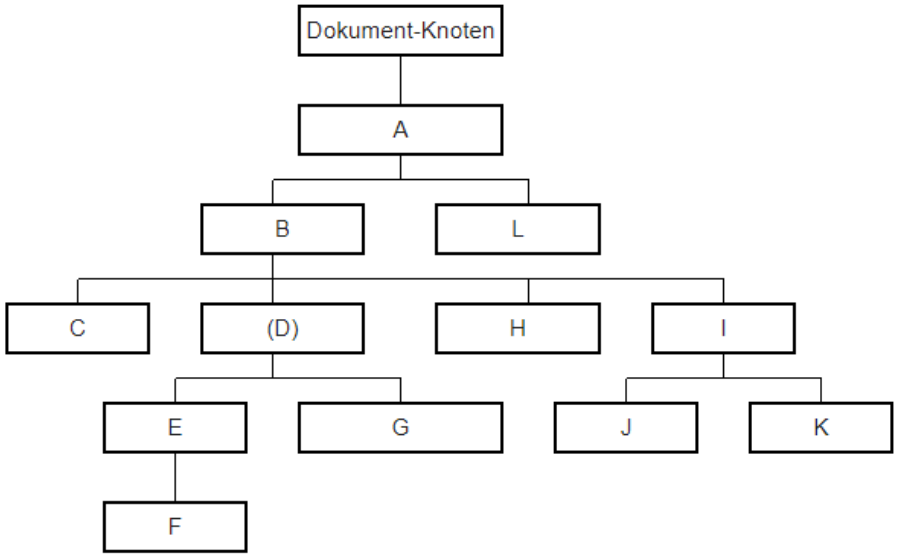
Ziel	CSS-Selektor	XPath
All Elements	*	//*
All Elements: p	p	//p
All Child Elements of p	p >*	//p/*
All Elements with ID foo	#foo	//*[@id='foo']
All Elements with class foo	.foo	//*[contains(@class, 'foo')
All Elements with attribute title	*[title]	//*[@title]
First Child of all p	p>*:first-child	//p/*[0]
All Elements p with Child a	*Not possible*	//p[a]
Next Element	p + *	//p/follow-sibling::*[0]
Previous Element	*Not possible*	//p/preceding-sibling::*[0]

Chap. 4 XPaths

XPath

- XML Path Language (XPath) ist eine Abfragesprache, um Teile eines XML-Dokumentes zu adressieren und auszuwerten
- Seit HTML5 ist die Struktur von HTML-Dokumenten äquivalent zu XML-Dokumenten
- XPath-Ausdruck adressiert Teile eines XML-Dokuments, das dabei als Baum betrachtet wird

Schema eines XML-Dokuments



Quelle: Eigene Darstellung

XPath-Nodes

XPath-Nodes

In XPath gibt es sieben Arten von Knoten: element, attribute, text, namespace, processing-instruction, comment, and document nodes

XML-Dokumente werden als Bäume von nodes behandelt.

Das oberste Element des Baums wird als root-Element (Stammelement) bezeichnet.

XPath-Nodes

Beispiel

```
<?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>
```

- Root-Element node: <bookstore>
- Element node: <author>J K. Rowling</author>
- Attribute node: lang=”en”

Beziehungen zwischen Nodes: parent

Jedes Element und Attribut hat ein übergeordnetes Element (parent).
Im folgenden Beispiel ist das <book> element das übergeordnete Element von <title>, <author>, <year> und <price>:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Beziehungen zwischen Nodes: children

Nodes-Elemente können null, ein oder mehrere children (untergeordnete Knoten) haben.
Im folgenden Beispiel sind <title>, <author>, <year> und <price> alle untergeordnete Elemente des <book>-Elements:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Beziehungen zwischen Nodes: siblings

Nur Nodes mit dem selben parent (übergeordneten Element). Im folgenden Beispiel: <title>, <author>, <year> und <price>

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Beziehungen zwischen Nodes: ancestors

Alle parent-Elemente einer Node und deren parent.
Im folgenden Beispiel sind <book> und <bookstore> die ancestors von <title>

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Beziehungen zwischen Nodes: descendants

Alle child-Elemente einer Node und deren child
Im folgenden Beispiel sind <book>, <title>, <author>, <year> und price die descendants von <bookstore>

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Syntax zur Auswahl von Nodes

XPath verwendet Pfadausdrücke, um Nodes in einem XML-Dokument auszuwählen. Die Node wird ausgewählt, indem einem Pfad oder Schritten gefolgt wird. Hier die nützlichsten Pfadausdrücke:

Ausdruck	**Beschreibung**
nodename	Auswahl aller Nodes mit dem Namen "nodename"
/	Auswahl der root-Node
//	Auswahl von Nodes innerhalb der aktuellen Node, welche der Auswahl entspricht
.	Auswahl der aktuellen Node
..	Auswahl des parent der aktuellen Node
@	Auswahl von attributes

Prädikate zur Auswahl von Nodes

Prädikate werden verwendet, um eine bestimmte Node oder eine Node mit einen bestimmten Wert zu finden.
Prädikate werden immer in eckige Klammern gesetzt.

Pfadausdruck	**Beschreibung**
nodename[1]	Wählt das erste Element mit entsprechendem Namen aus
nodename[last()]	Wählt das letzte Element mit entsprechendem Namen aus
nodename[last()-1]	Wählt das vorletzte Element aus
nodename[position()<3]	Wählt die ersten beiden Element aus
nodename[@lang]	Wählt alle entsprechenden Elemente mit einem attribut lang aus
nodename[@lang='en']	Wählt alle entsprechenden Elemente aus, bei denen das attribute lang den Wert 'en' hat
nodename[price>35.00]	Wählt alle entsprechenden Elemente aus, die ein price Element mit einem Wert von größer 35 haben

Übung zu XPaths

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<dok>
    <!-- ein XML-Dokument -->
    <kap title="Nettes Kapitel">
        <pa>Ein Absatz</pa>
        <pa>Noch ein Absatz</pa>
        <pa>Und noch ein Absatz</pa>
        <pa>Nett, oder?</pa>
    </kap>
    <kap title="Zweites Kapitel">
        <pa>Ein Absatz</pa>
        <pa format="bold">Erste Zeile</pa>
        <pa format="bold">Zweite Zeile</pa>
        <pa format="italic">Dritte Zeile</pa>
    </kap>
</dok>
```

Chap. 5 Rvest

Was ist Rvest?

Das Packet *rvest* macht es einfach Daten von Webseiten (HTML & XML) zu extrahiert (web scraping) und zu manipulieren. Es wurde für die Arbeit mit dem Packet *magrittr* (Pipe-Operator: %>%) entwickelt, um die Syntax für gängiger Web-Scraping-Aufgaben zu vereinfachen

Grundlegende Funktionen

Nachdem einlesen eines HTML-Dokuments mit read_html():

- Auswahl eines Teils des Dokuments mittel 'CSS-Selectors': html_nodes()
- Extraktion von bestimmten Komponenten:
  - html_name() (Namen des tag)
  - html_text() (Gesamter Text im tag)
  - html_attr() (Ausprägung eines 'attribute')
  - html_attrs() (Alle 'attributes')
- Tabellen in einen Datensatz "parsen": html_table()
- Navigieren auf der Webseite: back(), forward()

Erste Schritte mit Rvest

Webseite einlesen und filtern

```r
#Install and load the package/library
  if(!require("rvest")) install.packages("rvest")
  library(rvest)

#Start by reading a HTML page
  starwars <- read_html("https://rvest.tidyverse.org/articles/starwars.html")

#Get an overview of the structure
  films <- starwars %>% html_nodes("section")
  films

#Extract the movies' names /extract one element per film
  title <- films %>%
    html_node("h2") %>%
    html_text(., trim = TRUE)
  title

episode <- films %>%
  html_node("h2") %>%
  html_attr("data-id") %>%
  as.numeric()
episode
```

Tabellen einlesen

Wenn die Seite tabellarische Daten enthält, können diese mit html_table() direkt in einen Datensatz konvertiert werden.

```
#Start by reading a HTML page
  html <- read_html("https://en.wikipedia.org/w/index.php?
                    title=The_Lego_Movie&oldid=998422565")

#extract table
  html %>%
  html_node(".tracklist") %>%
  html_table()
```

Chap. 6 RSelenium

Why RSelenium

Wie in den vorherigen Abschnitten erwähnt, könnte auf JavaScript-Websites der Großteil des Inhalts mit JavaScript generiert werden. Wenn Sie einen Browser verwenden, werden beim Laden der Originalseite mehrere Anfragen an Sie gestellt. Wenn Sie jedoch rvest verwenden, wird nur die ursprüngliche Seite geladen und JavaScript wird nicht ausgeführt. Daher sind einige Daten nur im Browser verfügbar.
RSelenium verwendet den Selenium Web Driver, der einen Browser (normalerweise Chrome und Firefox) simuliert und Webseiten automatisch rendert (Umgewandlung des Codes in visuelle Darstellung). Er führt alle JavaScript-Codes für Sie aus, sodass Sie eine analysierte Seitenquelle anstelle einer nicht analysierten (rohen) erhalten.

Pros von RSelenium

Pro:

- Eines der größten Probleme beim Web-Scraping ist, dass das, was Sie im Browser sehen und was Sie als Antwort erhalten, unterschiedlich ist. Mit RSelenium können Sie dieses Problem vermeiden
- RSelenium → Rendert von JavaScript generierten Inhalt automatisch

Cons von RSelenium

Cons:

- Sehr langsam: Da der Browser alles auf der Webseite lädt (einschließlich Fotos, Anzeigen oder sogar Videos), ist er im Vergleich zu Anfragen mit httr oder rvest sehr langsam
- Fehlende Unterstützung: Im Vergleich zu Selenium in Python hat RSelenium keine große Benutzerbasis und daher keine Unterstützung. Wenn Sie jemals nach RSelenium-bezogenen Fragen gesucht haben, haben Sie möglicherweise bereits Folgendes herausgefunden: Viele Lösungen könnten veraltet sein, Sie konnten keine verwandten Themen finden oder die einzigen verfügbaren Lösungen waren für Python.

Interaktion mit dem Webdriver in Rselenium

Initialisierung

```
#Install and load the package/library
  if(!require("RSelenium")) install.packages("RSelenium")
  library(RSelenium)

# start the server and browser(you can use other browsers here)
  rD <- rsDriver(browser=c("firefox"))
  driver <- rD[["client"]]

# navigate to an URL
  driver$navigate("http://books.toscrape.com/")

#close the driver
  driver$close()

#close the server
  rD[["server"]]$stop()
```

RSelenium navigieren

URL öffnen:

```
driver$navigate("http://books.toscrape.com/")
```

Element anklicken:

```
# navigate to an URL
  driver$navigate("http://toscrape.com/")

# find the element
  elements <- driver$findElements("a",using = "css")

# click the first link
  elements[[1]]$clickElement()
```

Vor und Zurück navigieren:

```
driver$goBack()
driver$goForward()
```

Weitere Aktionen

Hoch und Runter scrollen

```
driver$navigate("http://quotes.toscrape.com/scroll")

# find the webpage body
  element <- driver$findElement("css", "body")

#scroll down once ----
  element$sendKeysToElement(list(key = "page_down"))
```

Öfter scrollen

```
element <- driver$findElement("css", "body")

# Scroll down 10 times
  for(i in 1:10){
      element$sendKeysToElement(list("key"="page_down"))
      # please make sure to sleep a couple of seconds to since it takes time to load contents
      Sys.sleep(2)
  }
```

Klicks simulieren

Einzelnes Element anklicken

```
#locate element using CSS(find the first match)
  driver$navigate("https://scrapethissite.com/pages/ajax-javascript/#2011")
  element <- driver$findElement(using = "css",".year-link")
  element$clickElement()
```

Mehrere Elemente anklicken

```
  driver$navigate("https://scrapethissite.com/pages/ajax-javascript/#2011")
  elements <- driver$findElements(using = "css",".year-link")
  for(element in elements){
      element$clickElement()
      Sys.sleep(2)
  }
```

Texteingabe simulieren

Text eingeben und suchen

```
  driver$navigate("https://www.google.com/")

#select input box
  element <- driver$findElement(using = "css",'input[name="q"]')
#send text to input box. don't forget to use `list()` when sending text
  element$sendKeysToElement(list("Web Scraping"))

#select search button
  element <- driver$findElement(using = "css",'input[name="btnK"]')
  element$clickElement()
```

Input Box säubern

```
  driver$navigate("https://www.google.com/")

#selcet input box
  element <- driver$findElement(using = "css",'input[name="q"]')
  element$sendKeysToElement(list("Web Scraping"))

#clear input box
  element$clearElement()
```

Chap. 7 Projekte

Gruppenarbeit

Bilden Sie Gruppen in der Größe von 2 bis 4 Personen, wählen Sie eine interessierende Fragestellung aus und versuchen Sie die nötigen Daten zur Bearbeitung der Fragestelle per Web Scraping in einen Datensatz in R zu überführen.

Chap. 8 Besprechung & Ausblick

Literatur

- RDocumentation: rvest
- Web Scraping Reference: Cheat Sheet for Web Scraping using R
- XPath Tutorial
- CSS Selector Reference

Gibt es noch weitere Fragen?