HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Lon-Capa Emulation mit Web Scraping

Nicolai Böttger

Bachelor-Arbeit im Studiengang „Angewandte Informatik“

27. Oktober 2020

Autor Nicolai Böttger
1476431
nboettger532@gmail.com

Erstprüferin: Prof. Dr. Frauke Sprengel
Abteilung Informatik, Fakultät IV
Hochschule Hannover
frauke.sprengel@hs-hannover.de

Zweitprüfer: Prof. Dr. Arne Koschel
Abteilung Informatik, Fakultät IV
Hochschule Hannover
arne.koschel@hs-hannover.de

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Bachelor-Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, den 27. Oktober 2020 Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Einführung

Lon-Capa[1] ist eine Lernplattform, die an der Hochschule Hannover eingesetzt wird. Auf dieser können Dokumente, wie Vorlesungen, veröffentlicht und Übungsaufgaben im Bereich der Mathematik erstellt werden. Eine weitere Lernplattform, die für die meisten anderen Module genutzt wird, ist *Moodle*. Diese eignet sich besser, um mehrere und verschiedene Vorlesungen zu gruppieren. Da für beide Systeme die gleichen Login-Daten verwendet werden, wäre es sinnvoll eine Möglichkeit zu finden, mit der man aus einem der beiden Systeme direkt auf das jeweils andere zugreifen könnte.

Lon-Capa wurde zum jetzigen Stand (Mai 2020) das letzte Mal vor 3 Jahren (Juni 2017) aktualisiert. Es ist ungewiss, ob und wann ein Release einer aktualisierten Version erscheint. Dies macht Lon-Capa zu einem potentiell guten Kandidaten, um als Legacy-System zu fungieren. Aus diesem könnten mithilfe von Web Scraping Tools Daten ausgelesen und neu aufbereitet werden, ohne das System zu verändern.

Ziel dieser Arbeit ist es, zu untersuchen, ob es möglich bzw. sinnvoll ist auf Lon-Capa mithilfe von Web Scraping Technologien zuzugreifen und welche sich am besten dafür eignen. Zusätzlich werden im Ausblick einige Ansätze diskutiert, wie ein mögliches Scraping Programm in ein System wie Moodle integriert werden könnte.

1.2 Gliederung

Diese Arbeit gliedert sich grundsätzlich in 5 Kapitel. Im ersten wird eine kleine Einleitung zum Thema der Bachelorarbeit, und was das Ziel dieser ist, aufgeführt.

Das 2. Kapitel erklärt Hintergrundinformationen über die allgemeinem Themen dieser Arbeit, also was Web Scraping ist und wie man einen Web Scraper umsetzen könnte. Außerdem wird auf die Struktur der -Website und erste Einschätzungen dessen Eignung für Web Scraping eingegangen.

[1] Lon-Capa - Abkürzung von **L**earning **O**nline **N**etwork with **C**omputer-**A**ssisted **P**ersonalized **A**pproach (`https://www.lon-capa.org/`)

In Kapitel 3 werden Kriterien zum Bewerten der in Kapitel 2 aufgeführten Umsetzungsmöglichkeiten aufgestellt und verwendet. Außerdem werden die wichtigen Funktionen von Lon-Capa analysiert und für die Nutzung durch ein Web Scraper Programm vorbereitet.

Kapitel 4 erläutert die Konzeption und Entwicklung eines möglichen Web Scrapers für Lon-Capa Daten. Dafür werden zuerst Anforderungen an das System aufgestellt und danach die Funktionsweise des entwickelten Programms erklärt. Zusätzlich werden verschiedene Scraping Ansätze untersucht, um mit dem Lon-Capa System zu interagieren

In Kapitel 5 werden die Ergebnisse aus Kapitel 4 erläutert und die Frage beantwortet, inwiefern es sinnvoll ist, Web Scraping für Lon-Capa zu verwenden. Außerdem wird evaluiert, ob die Entwicklung des Prototypen erfolgreich war oder warum und woran sie gescheitert ist.

Kapitel 6 fasst die Ergebnisse der vorherigen Kapitel nochmal zusammen und zieht ein Fazit über die Arbeit. Zusätzlich werden noch einige Ansätze für eine mögliche Integration des Programms in ein anderes System, welche auf die Arbeit aufbauen, aufgeführt.

1.3 Einordnung

Durch die permanente Entwicklung und Verbesserung der IT-Infrastruktur durch neue Technologien kann es schnell passieren, dass ältere, aber eigentlich noch funktionierende Software komplett neu ersetzt wird, anstatt sie an die neuen Technologien anzupassen. Mithilfe von Web oder Screen Scraping kann so ein älteres System weiterhin bestehen bleiben und dessen Funktionalität mit neuen Technologien erweitert werden. In dieser Arbeit wird unter anderem auf mögliche Kriterien eingegangen, die genutzt werden können, um zu beurteilen, ob sich das eigene System für eine Verarbeitung mithilfe von Web Scrapern anbietet.

2 Theoretischer Hintergrund

2.1 Definition Web Scraping

Die folgenden Unterkapitel definieren den Begriff Web Scraping und beschreiben Anwendungsgebiete sowie mögliche Probleme in der Benutzung von Web Scrapern.

2.1.1 Was ist Web Scraping

Screen Scraping (bzw. Web Scraping) bezeichnet den Prozess des Sammelns von Anzeigedaten aus einer Applikation zum Anzeigen dieser - in möglicherweise einer anderen Form - in einer anderen Applikation. Da es in dieser Arbeit um das Extrahieren von Daten aus einer Webseite geht, wird im Folgenden der Begriff Web Scraping verwendet. Beim Web Scraping wird nicht auf die dem System zu Grunde liegende Datenbank zugegriffen, sondern die Daten aus der Benutzeroberfläche (dem HTML-Code) ausgelesen. Man braucht also keinen direkten Zugang zu dem Server hinter der Anwendung, sondern „scrapet“ die Daten direkt aus der Anwendung. Der Scraper kommuniziert also wie ein normaler Nutzer mit dem zugrundeliegenden System und benötigt keine zusätzlichen Informationen über die Funktionsweise der Applikation.

2.1.2 Anwendungsbereiche von Web Scraping

Anwendung findet Web Scraping oft bei der Übertragung von Legacy-Systemen in Anwendungen mit neuen moderneren User-Interfaces. Normalerweise würde man ein veraltetes System durch ein neues ersetzen, was in älteren Firmen sehr viel Arbeit mit sich ziehen würde und teilweise gar nicht möglich wäre. Auch eine Erweiterung eines fertigen Systems könnte sich aufgrund von fehlender Dokumentation, Programmcode oder ähnlichem schwierig gestalten. So ein System könnte als Legacy System fungieren, was bedeutet, dass es in seiner fertigen funktionierenden Form nicht mehr direkt verändert wird, sondern als erweiterte Datenbank dient.

Mithilfe von Scraping Tools können dann die Output Daten des Legacy Systems neu aufbereitet und an eine andere Anwendung übergeben werden und das ohne die Notwendigkeit, den Funktionsprozess des Legacy-Systems zu verändern. Da zum Sammeln dieser Daten keine direkte Verbindung zur hinter liegenden Datenbank oder ähnlichem

notwendig ist, kann theoretisch auch jeder Benutzer, der nichts mit der Entwicklung des Systems an sich zu tun hatte, so ein Tool benutzen, um Daten abzufangen. Abgesehen von der Erneuerung von Legacy-Systemen wird Web Scraping zusätzlich benutzt um Daten aus mehreren verschiedenen Quellen in einer einzigen zusammenzufassen und anzuzeigen. So ist es beispielsweise möglich Internetseiten von verschiedensten Märkten zu durchsuchen und deren Angebote zu einem bestimmten Suchbegriff herauszufiltern.

2.1.3 Funktionsweise eines Web Scrapers

Ein Web Scraper kommuniziert mit der auszulesenden Webseite wie ein normaler Nutzer und hat in der Regel keine zusätzlichen Kenntnisse über diese außer ihren HTML[1]-Code. Grundsätzlich kann man die Aufgaben eines Scrapers in 3 Teile aufteilen. Zuallererst lädt der Scraper den Source Code der Webseite. Je nach Aufgabe des Scrapers kann dieser dann spezifische Informationen mithilfe von einem Selector[2] aus dem Code extrahieren und weiter verarbeiten. Dieser Selector beschreibt den „Weg“ um zu dem spezifischen Element zu gelangen und könnte z.B. ein regulärer Ausdruck oder ein CSS[3] Selector sein. Als letztes werden die ausgelesenen Daten dann noch in ein verarbeitbares Format wie z.B. XML[4] umgewandelt.

Allein durch dieses Auslesen der Daten ist eine Interaktion mit der zugrundeliegenden Seite aber noch nicht immer möglich. Zwar können beispielsweise Links zu anderen Seiten ausgelesen werden, allerdings können nicht direkt Buttons geklickt werden. Um solche Interaktionen zu ermöglichen, muss der Scraper entweder den POST-Request[5], welcher bei einem Klicken des Buttons ausgeführt wird, manuell abschicken oder mithilfe eines zusätzlichen Programms den Klick eines „echten“ Nutzers simulieren.

Abbildung 2.1: Funktionsweise eines Web Scrapers.

Weiterhin können Scraper nach Art ihrer Vorgehensweise aufgeteilt werden. Entweder lesen sie einmalig alle Informationen aus dem alten System und speichern sie in einer neuen Datenbank oder sie sammeln kontinuierlich Daten und fügen sie nach und nach in einer Datenbank ein. Ein Beispielalgorithmus für einen Scraper, der eine Seite nach

[1] Hypertext Markup Language
[2] Die Aufgabe und der Einsatz von Selektoren werden in 2.3 genauer beschrieben
[3] Cascading Style Sheets - Mechanismus zum Verändern des Aussehens von HTML-Elementen
[4] Extensible Markup Language - Dateiformat zur strukturierten Darstellung von Daten
[5] HTTP Anfragemethode zum Senden von Daten an den Server

allen URLs, die sich auf der Seite befinden und den gleichen Domänennamen besitzen, durchsucht und diese gefundenen Seiten rekursiv weiter durchsucht, könnte so aussehen [vgl. [Nas], Implemented Algorithm]:

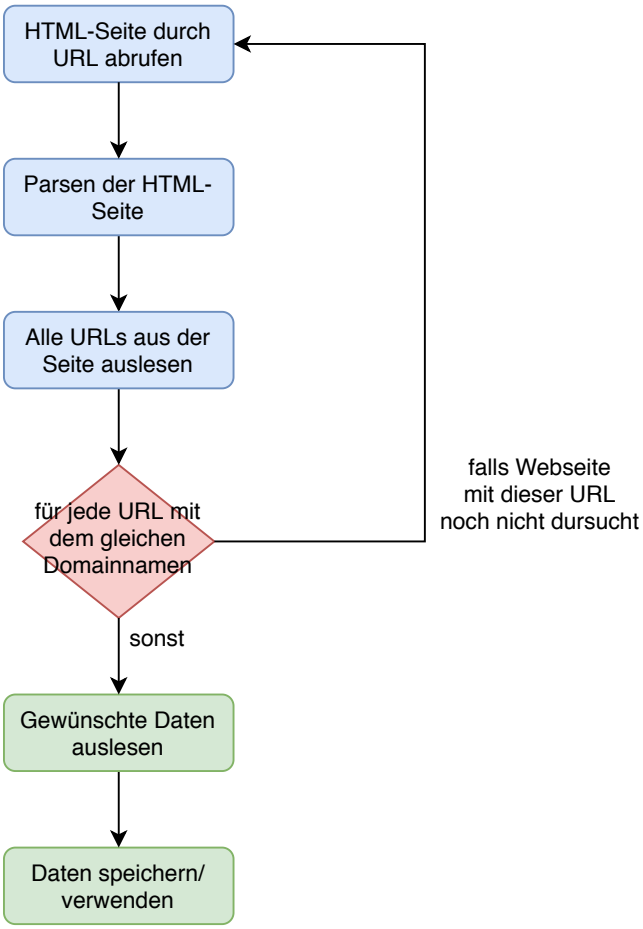


Abbildung 2.2: Web Scraper Beispielalgorithmus.

2.1.4 Probleme beim Web Scraping

Das wohl größte Problem von Web Scrapern ist die Instabilität dieser Anwendungen. Solange das System, auf welches sie zugreifen nicht verändert wird, sollten keine Probleme auftreten, jedoch kann die kleinste Veränderung der Struktur der Seite schon dazu führen, dass der Scraper nicht mehr funktioniert und somit überarbeitet werden muss. Wird ein Element beispielsweise durch einen CSS-Selector mithilfe der ID abgefragt, würde eine Veränderung der ID dazu führen, dass der Selector es nicht mehr findet. Oder wird eine auszulesende Tabelle in einem neuen Tag eingeschlossen, würde ein XPath-Selector[6] diesen nicht mehr finden. Ein weiteres Problem ist, dass ein Perfor-

[6]XML Path Language - Abfragesprache zum Addressieren von Teilen eines XML-Dokumentes

manceverlust beim Einsatz von Scraping-Software schon fast vorprogrammiert ist. Da der Scraper über die Nutzersicht auf das System zugreift, muss er auch mindestens solange warten, bis dieses alle Informationen vom Server bekommen hat. Nun muss dieser die Daten aber zusätzlich noch vom Client auslesen, was zwangsläufig zu einer längeren Wartezeit führt. Im besten Fall bleibt die Geschwindigkeit nahezu gleich, in der Regel muss aber mit etwas Geschwindigkeitsverlust gerechnet werden. Außerdem wirkt sich eine schlecht programmierte HTML-Struktur des Legacy-Systems auch auf den Scraper aus, welcher diese dementsprechend bearbeiten muss [vgl. [Gee], Challenges to Web Scraping].

Ein weiteres Problem ist, dass viele Webseitenbetreiber ihre Webseitendaten nicht von automatisierter Software ausgelesen haben wollen. Dafür gibt es verschiedene Anti-Scraping Technologien, welche beispielsweise dynamisch HTML Element-IDs ändern oder IPs von den jeweiligen Web Scrapern blockieren. Durch die vielen verschiedenen Requests, die ein Web Scraper an den Server sendet, ist er meist einfach von einem gewöhnlichen Nutzer unterscheidbar und kann so leicht geblockt werden. Zusätzlich können CAPTCHA[7] verwendet werden, um automatisierte Programme zu blockieren.

2.2 Umsetzungmöglichkeiten für Web Scraper

Es existieren viele verschiedene Möglichkeiten um einen Web Scraper umzusetzen. Welchen Ansatz man wählen sollte, hängt stark von dem zu „scrapenden“ System ab. In diesem Kapitel werden zuerst verschiedene Techniken vorgestellt, auf welche ein Scraper basieren könnte. Nachfolgend werden verschiedene Programmier-Frameworks und Libraries untersucht und analysiert. Danach werden noch einige fertige Tools betrachtet und am Ende beide Möglichkeiten verglichen.

2.2.1 Web Scraping Techniken

Die simpelste Form des Web Scrapings ist das manuelle "Copy-Pasten"der spezifischen Daten aus der Webseite. Da dies aber viel Arbeit mit sich bringt und für komplexe Webseiten nicht realistisch ist, wird dieser Ansatz in der Praxis kaum angewendet. Stattdessen werden automatisierte Verfahren erstellt, die die erforderlichen Daten automatisch und dynamisch auslesen. Die erste Möglichkeit spezifische Daten auszulesen ist mithilfe von *Text Pattern Matching*. Dabei wird mithilfe von regulären Ausdrücken und Unix-Befehlen wie "grep"[8] der HTML-Code der Seite durchlaufen und abgesucht. Eine weitere Technik ist das *HTML Parsing*. Dabei wird die HTML-Seite mithilfe von Parsern (meistens in Javascript) analysiert und in strukturierte Komponenten gegliedert, aus welchen

[7] Ein CAPTCHA ist ein Programm, das Websites vor Bots schützt, indem es Tests generiert und bewertet, die Menschen bestehen können, aber aktuelle Computerprogramme nicht. `http://www.captcha.net/`

[8] grep - UNIX-Befehl zum Suchen von Wörten oder Mustern in einer Textdatei

dann beispielsweise Links oder Texte extrahiert werden können. Als andere Methode existiert noch das *DOM-Parsing*, bei welchem Scraper mithilfe von DOM[9] Parsern die interne Struktur der Webseite analysieren können. Mithilfe von XPath Tools können diese XML Dokumente dann bearbeitet und durch Parameter bestimmte Knoten in der Datei ausgewählt werden [vgl. [Rad], Automated Scraping].

2.2.2 Web Scraping Libraries und Frameworks

Es existieren viele Web Scraping Libraries und Frameworks, welche speziell für diesen Anwendungszweck entwickelt wurden. Im Folgenden werden einige von ihnen vorgestellt und später im Bezug zur Struktur von Lon-Capa entschieden, welches sich am besten für die Entwicklung eines Scrapers eignet.

Eines der bekanntesten Frameworks ist das Python-Framework Scrapy[10]. Es ist ein Open-Source Projekt, welches zuerst im Jahre 2008 veröffentlicht wurde und bis heute noch weiterentwickelt wird. Es basiert auf einer Engine, welche den Datenfluss zwischen allen Komponenten des Frameworks managt, einem Downloader, welcher dafür verantwortlich ist, den Quelltext der Webseiten herunterzuladen, und einem Scheduler, welcher Aufgaben von der Engine enthält und diese zeitlich sortiert [vgl. [MM], Scrapy]. Der Scraping-Code wird vom Programmierer in sogenannten Spider-Klassen geschrieben und nach dem Herunterladen von der internen Item Pipeline weiterverarbeitet und gegebenenfalls in zum Beispiel einer Datenbank gespeichert.

Ein weiteres Open-Source Web Scraping Framework ist Web-Harvest[11]. Es ist in Java geschrieben und sowohl direkt in Java-Code als auch über die Kommandozeile oder auch über eine GUI-Oberfläche benutzbar. Die zu „scrapende“ Webseite wird hierbei zu Beginn heruntergeladen und danach von einem Parser von HTML zu XML umgewandelt. Danach sucht ein XPath Prozessor nach den benutzergewünschten Daten. Es ist demnach nicht möglich, mit Web-Harvest mithilfe von anderen Selektoren, wie z.B. einem CSS-Selektor, nach Daten zu filtern.

Neben den Standard-Frameworks, welche jeweils nur eine Webseite bearbeiten und die Daten an den Nutzer liefern, existiert noch der Selenium Web Driver[12]. Dieser simuliert im Gegensatz zu den anderen Frameworks einen „richtigen“ Benutzer und ermöglicht es, leicht mit interaktiven Webseiten zu interagieren. Dafür behält er durchgehend eine offene Instanz der zuletzt besuchten Webseite aktiv, auf die automatisiert und programmatisch zugegriffen werden kann. Dies ermöglicht auch das Klicken eines Buttons um z.B. ein HTML-Formular abzuschicken.

[9] Document Object Model - definiert Struktur und Inhalt von XML Dokumenten
[10] Scrapy - `https://scrapy.org/`
[11] Web-Harvest - `http://web-harvest.sourceforge.net/`
[12] Selenium Web Driver - `https://www.selenium.dev/documentation/en/webdriver/`

2.2.3 Web Scraping Tools

Neben Libraries und Frameworks gibt es auch viele fertige Web Scraping Tools, welche von Unternehmen angeboten werden. Auch hier werden im Folgenden einige von ihnen vorgestellt. Das erste Tool ist Visual Web Ripper[13], welches von Sequentum in 2006 entwickelt wurde. Es bietet die Möglichkeit, alle möglichen Arten von Webseiten wie z.B. E-Commerce oder Forums zu bearbeiten und Daten von diesen zu sammeln. Außerdem könnten die Daten in vielen Formaten, wie z.B. CSV, Excel oder XML exportiert werden. Allerdings wird diese Software nicht mehr weiterentwickelt und Ende 2020 nicht mehr offiziell unterstützt [vgl. [S S]].

Ein weiteres Tool ist Screen-scraper[14], mit welchem man auch die Möglichkeit hat, eigene Skripte anzufügen. Es gibt 3 verschiedene Versionen für verschiedene Anwendungsbereiche und unterschiedliche Komplexität des Programms. Allerdings hat es den Nachteil, dass es für unerfahrene Nutzer schwer zu erlernen ist, da einiges an Vorwissen benötigt wird.

Mozenda[15] kann zusätzlich zum normalen Web Scraping auch noch Daten und Informationen aus PDF Dateien extrahieren und verarbeiten. Es gilt als leicht bedienbar, da es auf einem einfachen „Point-and-Click“ Interface basiert und so nicht viel technisches Wissen zur Bedienung benötigt wird. Zusätzlich kann ein Scraping Projekt direkt in der Mozenda Cloud Umgebung betrieben werden. Allerdings ist dieses Programm nach einer 30-tägigen Testphase kostenpflichtig.

2.2.4 Fazit

Welches Tool man zur Umsetzung seines Web Scrapers verwendet kommt, ganz auf die Seite an, die man „scrapen“ möchte sowie die eigene technische Kompetenz in diesem Gebiet. Möchte man lediglich einige frei zugängliche Daten von einer einzelnen Seite sammeln, reichen z.B. auch einfache Module, die lediglich den Quelltext der Seite herunterladen und nach Mustern absuchen. Sind das Ziel hingegen interaktive und dynamische Webseiten müssen Frameworks, wie die vorgestellten, verwendet werden.

2.3 Selektoren

Selektoren sind ein Mittel, um Elemente innerhalb von HTML-Dateien eindeutig zu identifizieren. Die 2 Selektoren, die dafür am meisten verwendet werden, sind CSS Selektoren

[13]Visual Web Ripper - `http://visualwebripper.com/`
[14]Screen-scraper - `https://www.screen-scraper.com`
[15]Mozenda - `https://mozenda.com`

und XPath. CSS Selektoren benutzen dabei die CSS Style Attribute der einzelnen Elemente, um sie zu identifizieren, während XPath Pfadausdrücke verwendet, um durch den DOM-Baum zu navigieren.

2.3.1 XPath

XPath steht für XML Path Language und ist eine Abfragesprache, um einzelne Knoten in XML-Dokumenten zu selektieren, kann also nicht nur für HTML-Dateien verwendet werden. Dafür werden Ausdrücke benutzt, mit denen das XML Dokument vom Wurzelknoten bis zum ausgewählten Element durchlaufen werden kann. Zur Verdeutlichung der Funktionsweise eines XPath Selektors betrachten wird diesen HTML-Code.

```
1  <html>
2      <head></head>
3      <body>
4          <div class="Einleitung">
5              <p id="Überschrift">Überschrift</p>
6              <p>Text...</p>
7          </div>
8          <div>
9              <a href="https://google.com">Google</a>
10         </div>
11         <a href="https://hs-hannover.de">Hochschule Hannover</a>
12     </body>
13 </html>
```

Abbildung 2.3: HTML Beispiel für XPath.

Diese Seite besteht außer aus den HTML-Standardelementen nur aus zwei Div Elementen, welche zum einen zwei Paragraph Elemente und zum anderen einen Link zur Seite Google enthalten und einen weiteren Link zur Webseite der Hochschule Hannover. Im Folgenden wird des öfteren von Eltern-und Kindelementen gesprochen. Mit Elternelement ist dabei das Element gemeint, welches das ausgewählte Element umschließt. In diesem Beispiel wäre das Element das Eltern-Element der beiden Paragraph Elemente und ein Kindelement des <body> Elements.
Möchte man nun alle Elemente in diesem Dokument selektieren, kann man dies mit dem Selektor `//*` erreichen.

```
> $x("//*")
< ▶ (9) [html, head, body, div.Einleitung, p#Überschrift, p, div, a, a]
```

Das **$x([...])**, welches den Selektor umschließt, ist eine Funktion der Chrome DevTools, mit der man seinen Selektorcode validieren kann. Rückgabe dieses Ausdruckes sind die

neun Elemente, welche in der HTML-Datei existieren. Wie aus CSS bekannt, werden Elementklassen mittels eines **.** und Elementids mit einem # gekennzeichnet.

```
> $x("//html/body/div/p")
< ▶ (2) [p#Überschrift, p]
```

Dieser Ausdruck liefert alle <p>-Elemente in diesem Pfad als Array zurück. Um ein spezifisches Paragraph-Element zu selektieren, muss ein Arrayzugriff mittels [**x**] nach dem **p** angefügt werden. Um nun ein verschachteltes Element alleine auszuwählen, muss der Pfad zu diesem Element als Selektorausdruck verwendet werden. Anstelle der Benutzung eines Positionsindexes kann, sofern es existiert, auch ein Attribut zur Identifizierung des gesuchten Elements verwendet werden. Dieses wird in der Form [**@Attribut="Wert"**] an das jeweilige Attribut angefügt.

```
> $x("//html/body/div/p[@id='Überschrift']")
< ▶ [p#Überschrift]
```

Möchte man alle Elemente eines bestimmten Typs in der HTML-Datei finden, kann man dies erreichen, indem man einfach den Namen des Elements nach dem // angibt.

```
> $x("//a")
< ▶ (2) [a, a]
```

Um nur den Freitext eines bestimmten Elements zu selektieren, kann am Ende der Angabe des Pfades ein **/text()** angefügt werden.

```
> $x("/html/body/div[2]/a/text()")
< ▶ [text]
```

In dieser Textrückgabe befindet sich nun der Text „Google“. Der Unterschied zwischen / und // ist, dass bei ersterem nur in den unmittelbaren Kindknoten und beim letzteren rekursiv auch in den Kindern des Elternknotens nach dem Element gesucht wird.

Dies sind die Grundfunktionen der XPath Language, mit denen man die meisten Elemente eines HTML-Dokuments identifizieren kann. Es existieren noch eine Menge weiterer Funktionen, mit denen man beispielsweise direkt den Text eines Elements als String zurückgeben oder mehrere Selektorstatements miteinander verknüpfen kann. Alle Funktionen können auf der W3Schools Website[16] und den darauffolgenden Seiten eingesehen werden.

[16] W3Schools (XPath) - `https://www.w3schools.com/xml/xpath_intro.asp`

2.3.2 CSS-Selektor

Heutzutage sind die meisten HTML-Seiten mittels CSS visuell verändert. Die angepassten Elemente können mittels CSS-Selektoren identifiziert und lokalisiert werden. Anstatt die Baumstruktur des DOM anhand der Pfade zu durchlaufen, wie bei XPath, existieren beim CSS-Selektor verschiedene Arten von Selektoren, mit denen man spezifisch nach den Eigenschaften der Elemente suchen kann. Die einfachen Selektoren suchen nach den HTML-Elementen mithilfe derer Attribute, wie der ID und der Klasse. Neben diesen einfachen Selektoren existieren auch noch Kombinatoren, mit welchen man Elemente in Bezug zu ihren relativen Positionen mit anderen Elementen lokalisieren kann. Als letztes gibt es noch Pseudoselektoren, welche es ermöglichen, durch Zusatzfunktionen wie z.B. „nth-child(x)“ genauer nach bestimmten Elementen zu suchen. Die nachfolgenden Selektorbeispiele beziehen sich wieder auf den HTML-Code aus dem XPath Kapitel.

Mithilfe eines einfachen „*“ ist es möglich, alle Elemente des HTML-Dokuments zu selektieren.

```
> $$("*")
◂ ▸(9) [html, head, body, div.Einleitung, p#Überschrift, p, div, a, a]
```

Nach Elementen kann einfach mit dem Namen des Elementes gesucht werden

```
> $$("p")
◂ ▸(2) [p#Überschrift, p]
```

Es ist auch möglich, nach einer ID oder einer Klasse direkt zu suchen, wodurch alle Elemente mit dieser ID oder der Klasse zurückgegeben werden.

```
> $$("#Überschrift")
◂ ▸[p#Überschrift]
```

Funktionen der Pseudoselektor-Klasse werden immer im Format **:Funktion** an den letzten Elementausdruck des Selektors angefügt.

```
> $$("p:last-child")
◂ ▸[p]
```

Mithilfe der Funktion **:last-child** kann das letzte Element einer Elementgruppe selektiert werden.

```
> $$("div:first-of-type")
◂ ▸[div.Einleitung]
```

Der Pseudoselektor **:first-of-type** selektiert immer das erste vorkommende Element eines Elementtyps.

Um die Funktionsweise von Kombinatoren zu verdeutlichen, wird eine neue HTML-Datei betrachtet.

```
1  <html>
2      <head></head>
3      <body>
4          <table>
5              <tr>
6                  <th>A</th>
7                  <th>B</th>
8                  <th>C</th>
9              </tr>
10             <tr>
11                 <td class="important">A1</td>
12                 <td id="a2">A2</td>
13                 <td>A3</td>
14             </tr>
15             <tr>
16                 <td class="important">B1</td>
17                 <td id="b2">B2</td>
18                 <td>B3</td>
19             </tr>
20         </table>
21     </body>
22 </html>
```

Abbildung 2.4: HTML-Tabelle Beispiel für CSS-Selector.

Das einzige Konstrukt der Seite ist eine Tabelle mit drei Tabellenreihen und drei Tabellenspalten mit bestimmten ID und Klassen-Attributen. Um alle Elemente, welche Kinderelemente eines bestimmten Elements sind, zu selektieren, benutzt man den Selektor **(Elternelement Ausgewähltes-Element)**

```
> $$("tr td")
< ▶ (6) [td.important, td#a2, td, td.important, td#b2, td]
```

Rückgabe dieser Abfrage sind alle <td>-Elemente, die Kinder von <tr>-Elementen sind.

```
> $$(".important + td")
< ▶ (2) [td#a2, td#b2]
```

Mithilfe des „+“ Operators werden Elemente, die direkt einem anderen Element folgen, selektiert. Rückgabe in diesem Beispiel sind alle Tabellenspalten, welche Elementen der

Klasse „important“ folgen. Neben diesen Kombinatoren existieren noch weitere, wie z.B. ein „,“, welches benutzt wird, um beide Elemente zu selektieren und ein „>“ im Format **(A > B)**, durch welches alle B-Elemente selektiert werden, die A-Elemente als direkte Elternelemente haben.

Auch dies sind nur einige der wichtigsten Kombinator-und Pseudoselektorfunktionen von CSS-Selektoren. Eine komplette Auflistung der Funktionen ist auf der SELFHTML[17] Website zu finden.

Abgesehen von den Unterschieden in der Syntax existiert zwischen beiden Ansätzen kein signifikanter Unterschied [vgl. [Tes]]. In dieser Arbeit werden hauptsächlich CSS-Selektoren zum Selektieren von Elementen der Lon-Capa Webseite verwendet, da sie in der Regel einfacher zu lesen sind.

2.4 Erläuterung Lon-Capa Aufbau

2.4.1 Struktur der Webseite

Um zu überprüfen ob Lon-Capa sich für Web Scraping eignet, wird an dieser Stelle auf die Struktur der Lon-Capa Webseite eingegangen. Hierbei werden nur die essentiellen Funktionen von Lon-Capa betrachtet, welche notwendig, sind um als Student mit der Seite zu arbeiten.

Ruft man die Seite über den Subdomänen-Link der Hochschule Hannover (`https://loncapa.hs-hannover.de`) auf, wird man direkt in ein Unterverzeichnis (`https://loncapa.hs-hannover.de/adm/roles`) weitergeleitet. Hat man sich vorher noch nicht eingeloggt, beziehungsweise nach Benutzung wieder ausgeloggt und befindet sich dementsprechend in keiner gespeicherten Sitzung, sieht man nun das Login-Portal für die Webseite. Das Login-Fenster besteht aus einem <form>-Element mit zwei <input>-Elementen, in welche der Benutzer seine Benutzerkennung und sein Passwort eintragen muss. Beide Elemente haben jeweils eine eindeutige ID und einen Namen, wobei das Feld für das Passwort am Ende seines Namens und der ID noch eine zufällig generierte Zahlenfolge hat, welche sich nach jedem Aktualisieren der Seite verändert. Zusätzlich zu diesen Feldern gibt es noch ein weiteres <input>-Element mit dem Bezeichner ”Domäne”, welches allerdings schon automatisch beim Aufrufen der Seite ausgefüllt ist. Durch den „Einloggen“-Button wird das Formular mit den jeweiligen Werten abgeschickt, und bei validen Daten wird der Benutzer auf eine neue Seite weitergeleitet.

Auf dieser befinden sich die Kurse, in denen der Benutzer eingeschrieben ist. Beim Klick auf den „Auswählen“-Button neben den einzelnen Kursen wird eine Javascript-Methode ausgeführt, welche auf eine neue Seite mit dem Inhaltsverzeichnis des Kurses weiterleitet. Hier werden die einzelnen Kapitel in einer HTML-Tabelle dargestellt. Beim Klick

[17] SELFHTML (Selektoren) - `https://wiki.selfhtml.org/wiki/CSS/Selektoren`

auf jeweils ein Kapitel wird mittels eines <a href>-Elements auf eine neue Seite weitergeleitet, welche die Unterkapitel des jeweiligen Kapitels anzeigt. Dort befinden sich dann entweder weitere Ordner mit bspw. den Vorlesungsfolien, welche sich bei einem Aufruf genauso verhalten, oder Übungen zu den einzelnen Themenfeldern. Beim wiederholten Klicken der Unterkapitel öffnet sich wieder die vorherige Seite. Beim Klicken der Vorlesungsfolien und Übungsblätter öffnen sich diese PDF-Dateien auf einer neuen Lon-Capa Seite. Abgesehen von den Seiten mit den direkten Übungen innerhalb von Lon-Capa sind diese beschriebenen Funktionen die essentiellen zur Navigation innerhalb der Seite.

Die Übungsseiten bestehen wieder aus einem <form>-Element, welches je nach Aufgabentyp andere <input>-Elemente besitzt. Am Anfang des Formulars befindet sich immer die Aufgabenstellung mit eventuellen Hinweisen zu den Aufgaben. Darunter sind dann die jeweiligen Input-Felder, welche der Benutzer ausfüllen muss. Diese sind je nach Aufgabentyp entweder ein Dropdown-Menü, ein Freitext-Feld, in welches der Benutzer seine Lösung der Aufgabe eintragen muss, oder Radio-Buttons, welche ausgewählt werden müssen. Das Dropdown-Menü ist ein <select>-Element, welches die jeweiligen Antwortmöglichkeiten als <option>-Element beinhaltet. Bei jeder Änderung der Auswahl wird eine Javascript-Methode aufgerufen, welche eine Variable innerhalb der Seite verändert und den Wert der ausgewählten Antwort in dieser speichert. Bei Aufgaben mit Freitextfeldern als Antwortmöglichkeit wird die gleiche Javascript-Methode bei jedem Eingeben oder Entfernen eines Buchstabens oder einer Zahl (etc.) aufgerufen. Auch hier wird die Eingabe dann in einer internen Variable gespeichert. Jede Antwortmöglichkeit hat einen Namen in der Form „HWVAL_...“, wobei ... je nach Anzahl der einzelnen Aufgaben durch eine Zahl ersetzt wird. Abgeschickt wird das Formular mit Klick auf den „Antwort einreichen“-Button, wonach die Antworten von dem Server überprüft werden. Nach einer Aktualisierung der Seite existiert ein neues Element, welches anzeigt, ob die abgegebene Antwort korrekt oder falsch war.

Diese Funktionen decken die Grundfunktionen von Lon-Capa ab und werden benötigt, um mit dem System zu arbeiten. Offensichtlich existieren noch weitere wichtige Funktionen, wie die Möglichkeit, Kommentare zu einzelnen Aufgaben zu schreiben und so Fragen zu stellen oder die Möglichkeit, die Anzahl seiner erreichten Punkte einzusehen, welche aber nicht zwingend benötigt werden, um Lon-Capa zu benutzen.

2.4.2 Eignung für Web Scraping

An dieser Stelle werden kurz erste Eindrücke erläutert, welche sich aus der Struktur folgern lassen, inwiefern Lon-Capa zur Anwendung von Web Scraping geeignet ist. Ein erstes mögliches Problem ist hierbei schon ganz zu Anfang zu erkennen. Web Scraping Tools benötigen eindeutige Identifikatoren, um HTML-Elemente im HTML-Quelltext zu lokalisieren und zu benutzen. Der Fakt, dass das Passwort-Element bei jeder Aktualisierung der Seite eine unterschiedliche ID und einen anderen Namen hat, erschwert

das Lokalisieren dieses Elements. Ein weiteres Problem ist außerdem, dass es bei Seitenwechseln innerhalb von Lon-Capa oft eine gewisse Initialisierungszeit gibt, bevor die Seite fertig geladen ist. Hier muss geprüft werden, wann die Seite wirklich fertig geladen hat, bevor man den Quelltext der Seite zum weiteren Verarbeiten durch einen Scraper herunterlädt.

Die Navigation innerhalb der Seite eignet sich allerdings gut für diesen Anwendungszweck, da sie fast ausschließlich über <a href>-Elementen, aus welchen man die Links zu den weiterführenden Seiten sehr leicht extrahieren kann und ohne viel Javascript-Code, funktioniert. Auch das Öffnen von Vorlesungsdateien und Übungblättern geschieht über diese Elemente und kann somit gut von einem Scraper umgesetzt werden. Problematisch wird Web Scraping allerdings, wenn der Scraper viel mit dem Clienten interagieren und Javascript ausführen muss. Genau dies ist bei der Auswahl des Kurses der Fall, in dem der Nutzer arbeiten möchte. Für diese Funktion müssen Javascript-Methoden aufgerufen werden, was die meisten Web Scraping Tools allerdings nicht ohne weiteres können.

Für das Abschicken der Übungen auf den Aufgabenseiten, zur Prüfung an den Server, muss ein Request, welcher normalerweise durch das Betätigen des „Antwort einreichen“-Buttons abgeschickt wird, manuell von dem Scraper verschickt werden. Dazu muss der Inhalt dieses Requests analysiert und überprüft werden, welche Daten an den Server geschickt werden. Viele Web Scraping Tools oder Frameworks bieten hierbei die Möglichkeit, einen solchen Request über Methodenaufrufe durchzuführen.

3 Analyse von LON-CAPA

3.1 Struktur der Arbeit

Im Folgenden wird kurz die Vorgehensweise, die in dieser Arbeit gewählt wurde, erläutert. Der größte Teil der Arbeit beschäftigt sich damit, ob und wie sich die Webseite LON-CAPA für Web Scraping eignet. Um dies zu untersuchen, wird auf praktische Weise versucht, ein Programm zu programmieren, mit dem es möglich ist, LON-CAPA zu steuern. Dafür werden zunächst einige Anforderungen aufgestellt, welche das Programm erfüllen muss. Im nächsten Schritt wird überprüft, ob die notwendigen Elemente, die zur Erfüllung dieser Anforderungen benötigt werden, mittels eines geeigneten Tools auslesbar sind und ob mit ihnen interagiert werden kann. Welches Tool sich am besten für diese Entwicklung eignet, wird anhand der Auswertungen der Struktur von LON-CAPA aus *Unterabschnitt 2.4.1* und *Unterabschnitt 2.4.2* erläutert und entschieden.

3.2 Bewertung der Umsetzungsmöglichkeiten

Um die Eignung der in *Unterabschnitt 2.2.2* und *Unterabschnitt 2.2.3* vorgestellten Frameworks und Tools zur Bearbeitung von LON-CAPA festzustellen, werden einige Kriterien aufgestellt, anhand welcher ihre Eigenschaften und ihre Kompatibilität mit LON-CAPA bewertet werden.

3.2.1 Kriterien

Das erste Kriterium, welches hierfür aufgestellt wird, ist das Kriterium **Anwendungsbereich**. Hier wird untersucht, für welche Art von Aufgabe das jeweilige Tool ursprünglich erstellt wurde und wo dessen Grenzen sind. Ist es beispielsweise nur möglich, Daten aus einer statischen Webseite zu laden und in einem bestimmten Datenformat zu speichern, oder kann das Tool auch mit dynamischen Webseiten arbeiten? Ein Tool wird hierbei als gut bewertet, wenn es möglichst viele Arten von Webseiten bearbeiten kann.

Ein weiteres Kriterium, nach dem entschieden wird, ist das Kriterium **Kosten**. Hierbei wird betrachtet, welche Kosten für das Benutzen des jeweiligen Tools anfallen. Kann man

es beispielsweise nur wenige Tage als Testversion benutzen, oder gibt es unterschiedliche Versionen des Tools, welche wiederum unterschiedliche Features bieten?

Um zu überprüfen, wie gut das jeweilige Tool an eigene Programmierungen anzubinden ist, wird das Kriterium **Erweiterbarkeit** benutzt. Da es nicht nur notwendig ist, die Daten einfach aus der Webseite auszulesen, sondern auch in einem Format auszugeben, in welchem mit den Daten auf programmatischer Ebene weitergearbeitet werden kann, ist es notwendig zu überprüfen, wie diese Daten ausgegeben werden. Zusätzlich muss überprüft werden, inwiefern das jeweilige Tool selbst von externem Programmcode gesteuert werden kann, da dieser dem Tool mitteilen muss, welche Elemente es selektieren soll. Ein Tool erfüllt dieses Kriterium vollständig, wenn es Daten in einem Format ausgibt, mit welchem das Programm arbeiten kann, ohne sie vorher in ein anderes Format umwandeln zu müssen und wenn alle Funktionen des Tools programmatisch steuerbar sind.

Die letzten Ansprüche an das Tool werden unter dem Kriterium **Dokumentation und Aktualität** zusammengefasst. Hierbei wird zum einen untersucht, wie umfangreich die Dokumentation ist, die für das jeweilige Tool existiert und zum anderen, ob das Tool noch weiterentwickelt wird. Zusätzlich wird, falls Informationen vorhanden sind, betrachtet, wie weit verbreitet das jeweilige Tool ist und wie viele Projekte mit diesem momentan entwickelt werden.

Anhand dieser vier Kriterien wird im Anschluss erläutert, welches Tool oder Framework und warum für die letztendliche Entwicklung des Scraping-Programms verwendet wird.

3.2.2 Bewertung

Da im Rahmen dieser Arbeit nicht alle vorgestellten Tools ausführlich getestet werden konnten, werden die Informationen, die zur Bewertung der jeweiligen Programme benötigt werden, hauptsächlich aus Literaturquellen genommen. Zuerst werden für die Bewertung die fertigen Tools und anschließend die vorgestellten Frameworks betrachtet.

3.2.3 Bewertung der Tools

Das erste vorgestellte Tool war der Visual Web Ripper. Dieser kann, ohne die Kriterien zu beachten, schon ausgeschlossen werden, da es ohne weiteres nicht mehr möglich ist, diese Software zu erwerben und ab 2020 der Support für die Anwendung komplett abgeschaltet wird. Für die Entwicklung einer neuen Anwendung eignet sich dieses Tool also nicht mehr.

Das nächste betrachtete Tool ist screen-scraper. Dieses Tool kann auch für komplexere Seiten verwendet werden. So ist es beispielsweise möglich, Links zu klicken, Text von einer Webseite zu kopieren und Daten in Formulare einzutragen und diese abzuschicken. Allerdings ist es mit der Standardversion nicht möglich, Cookies zu managen,

was für eine korrekte Navigation mit Lon-Capa nötig sein könnte. Bezüglich der Kosten lässt sich sagen, dass das Tool in drei Versionen existiert. Die Standardversion ist kostenlos, während die beiden anderen für jeweils immer mehr Features mehr kosten. Screen-scraper besitzt eine gute Erweiterbarkeit, da es möglich ist, Skripte in Programmiersprachen wie Python oder Java zu schreiben, welche mit dem Tool kommunizieren können. Eine ausführliche Dokumentation der einzelnen Funktionen des Tools existiert und ist durch Beispiele meistens leicht verständlich.

Damit lässt sich zusammenfassen, dass sich screen-scraper auf ersten Blick als Tool für die Emulation von Lon-Capa grundsätzlich eignet.

Das letzte Web-Scraping Tool, welches im Rahmen der Arbeit vorgestellt wurde, ist Mozenda. Mozenda bietet im Allgemeinen ähnlich viele Funktionen wie auch screen-scraper und bietet unter anderem noch die Möglichkeit, das entwickelte Programm in der Mozenda-Cloud direkt zu hosten. Allerdings muss, wie schon bei der Vorstellung des Programms erwähnt, nach einer 30-tägigen Testphase für das Tool bezahlt werden. Erweiterbar ist das Tool auch durch eigene Skripte in bspw. Python. Hierfür lassen sich auch einige Projekte auf GitHub finden, welche die Benutzung der Mozenda API in Python demonstrieren. Auch die Dokumentation des Tools ist übersichtlich und aktuell.

Zusammenfassend lässt sich auch hier sagen, dass sich auch Mozenda für die Entwicklung eines Web Scrapers für Lon-Capa eignet, allerdings ausscheidet, weil es kostenpflichtig ist.

3.2.4 Bewertung der Frameworks

Das erste Framework, welches untersucht wird, ist Scrapy. Grundsätzlich eignet sich Scrapy gut, um die meisten statischen Webseiten zu scrapen. Problematisch wird es, wenn die zugrundeliegende Seite viel mit Javascript arbeitet. Es ist beispielsweise sehr schwierig für traditionelle Scraping-Frameworks wie Scrapy Seiten zu bearbeiten, welche mithilfe von AJAX[1]-Calls Elemente dynamisch neu laden. Dies liegt daran, dass Scrapy den HTML-Quelltext mittels der URL sofort herunterlädt. Besitzt die Webseite nun einen Timer, welcher nach einiger Zeit Elemente mittels AJAX nachlädt, sind diese nicht im heruntergeladenen Quelltext vorhanden und können dementsprechend nicht selektiert werden. Da Lon-Capa, zumindest für die Kursauswahl der Website viel Javascript benutzt, erfüllt Scrapy dieses Kriterium nur bedingt. Da das Framework kostenlos ist, wird das Kriterium Kosten perfekt erfüllt. Auch das Kriterium Erweiterbarkeit ist durch den Fakt, dass Scrapy ein Framework für die Programmierung ist, erfüllt. Die Dokumentation ist durch viele Beispiele leicht verständlich, und es lassen sich viele Projekte finden, die Scrapy als Web-Scraping Software benutzen, was auf eine weite Verbreitung und Aktualität hinweist.

[1] Asynchronous JavaScript And XML (AJAX) - ermöglicht das asynchrone Aktualisieren einer Webseite durch Datenaustausch mit dem Webserver ohne die Seite erneut zu laden (`https://www.w3schools.com/xml/ajax_intro.asp`)

Da für Lon-Capa eine Software benötigt wird, die zum Teil auch mit Javascript-Funktionalitäten arbeiten können muss, eignet sich Scrapy nur bedingt als Scraping-Framework für die Website Lon-Capa.

Selenium funktioniert im Gegensatz zu den anderen Web-Scraping Frameworks grundlegend anders. Anstatt den Quelltext einfach nur hinterzuladen, wird mithilfe einer Browserinstanz die jeweilige Webseite direkt aufgerufen. Dies ermöglicht es auch, Javascript-Funktionen, die durch einen Button-Click ausgeführt werden würden, auszuführen. Allerdings verursacht diese Browserinstanz auch einen größeren Overhead als einfache Frameworks, die einfach nur den Quelltext herunterladen. Wie die anderen Frameworks auch erfüllt Selenium alle anderen Kriterien sehr gut.

Selenium eignet sich durch die vollständige Erfüllung aller Kriterien gut für die Emulation von Lon-Capa.

3.2.5 Fazit

Für die Entwicklung des Web-Scrapers für Lon-Capa wird das Framework Selenium benutzt, da es als einziges alle aufgestellten Kriterien gut erfüllt und sich so am besten eignet.

3.3 Scraping von Lon-Capa

Wie schon im Kapitel „Struktur der Webseite“ beschrieben, gelangt man beim ersten Aufruf der Lon-Capa Seite der Hochschule Hannover immer auf die Login-Seite. Um sich erfolgreich einzuloggen, ist es notwendig, die Benutzerkennung und das Passwort einzutragen und im Anschluss auf den Einloggen-Button zu klicken. Alle benötigten Elemente befinden sich in einem <form> Element.

3.3.1 Login-Seite

```
▼<form name="client" action onsubmit="return(send())">
    <input type="hidden" name="lextkey" value="-745139143">
    <input type="hidden" name="uextkey" value="-329338178">
  ▼<b>
      <label for="uname">Benutzerkennung</label>
      ":"
    </b>
    <br>
    <input type="text" name="uname" id="uname" size="15" value>
    <br>
  ▶<b>…</b>
    <br>
    <input type="password" name="upass1600614655" id=
    "upass1600614655" size="15">
    <br>
  ▶<b>…</b>
    <br>
    <input type="text" name="udom" id="udom" size="15" value="fh-
    hannover">
    <br>
    <input type="submit" value="Einloggen">
  </form>
```

Abbildung 3.1: Lon-Capa Loginseite HTML.

Das Input-Feld kann mit einem einfachen ID Selektor selektiert werden. Für das Passwort Input-Feld kann eine bestimmte Funktion der CSS-Selektoren benutzt werden. Während durch jedes Aktualisieren der Seite das Ende der ID des Elements verändert wird, bleibt der Anfang immer gleich. Durch das „^“ Symbol wird für jedes Element überprüft, ob es mit dem nachfolgenden Text beginnt.

```
> $$("#uname, [id^='upass']")
< ▶ (2) [input#uname, input#upass1600614655]
```

Mit diesem Selektor können also beide Elemente gefunden werden. Als letztes Element wird noch der Input Button zum Abschicken des Formulars benötigt. Dieses besitzt zwar kein ID Attribut, kann aber mittels seiner Position und des Typs selektiert werden.

```
> $$("[name='client'] [type='submit']")
< ▶ [input]
```

Diese Selektoren können so vom Selenium Webdriver benutzt werden, um den Benutzer seine Logindaten eingeben zu lassen und um danach das Login-Formular abzuschicken.

3.3.2 Kursauswahl-Seite

Weitergeleitet wird auf die Webseite, auf der es möglich ist, den Kurs auszuwählen, in dem der Benutzer arbeiten möchte. Die Kurse sind hier in einer Tabelle angeordnet und bestehen aus dem Feld zum Auswählen des Kurses, der Rolle des Benutzers in diesem Kurs, dem Namen des Kurses sowie dem Anfangs-und Enddatum des Kurses.

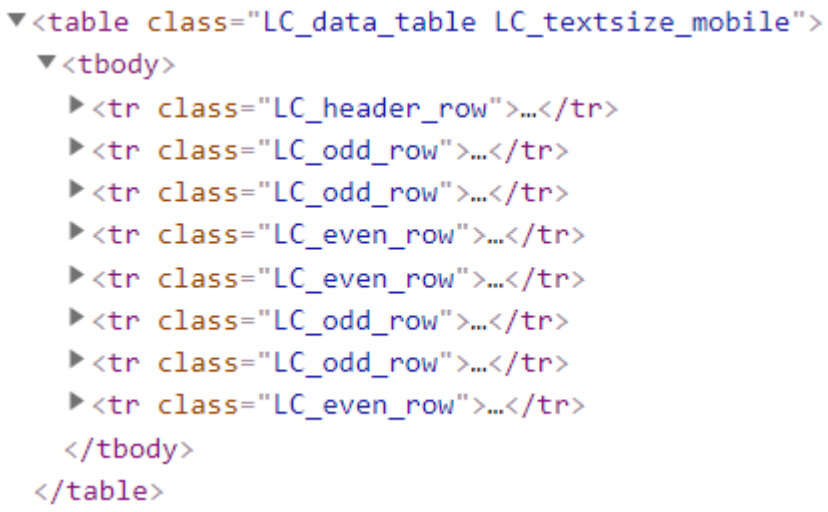


Abbildung 3.2: LON-CAPA Kursauswahlseite HTML.

Alle benötigten Elemente befinden sich hierbei in einem <table>-Element. Für jeden Kurs werden zwei <tr>-Elemente dynamisch generiert, wovon nur der erste immer die benötigten Daten enthält. In diesem Beispiel ist der Benutzer in drei Kurse eingeschrieben. Das erste <tr>-Element enthält die Headerinformationen für die einzelnen Spalten und das letzte, welcher Kurs derzeit ausgewählt ist. Die Elemente zwischen diesen sind die sechs Elemente für die drei Kurse. Von diesen sechs Elementen enthält das erste immer die jeweiligen Informationen und den Button zum Auswählen des Kurses. Um jeweils das erste Element zu selektieren, kann der Pseudoselektor *:nth-child(even)* verwendet werden, welcher nur gerade Elemente auswählt.

```
> $$("[class^='LC_data_table'] tr:nth-child(even)")
< ▶ (4) [tr.LC_odd_row, tr.LC_even_row, tr.LC_odd_row, tr.LC_even_row]
```

Dieser Selektor gibt die drei datentragenden Elemente und das letzte Element der Tabelle zurück. Innerhalb dieser Tabellenspalten befinden sich dann in <td>-Elementen die Kursbeschreibungen. Wird nun einer der Kurse ausgewählt, öffnet sich das Inhaltsverzeichnis des jeweiligen Kurses auf einer neuen Seite.

3.3.3 Inhaltsverzeichnis-Seite

Dieses Inhaltsverzeichnis zeigt nun alle eingetragenen Unterkapitel des jeweiligen Kurses in einer Tabelle an.

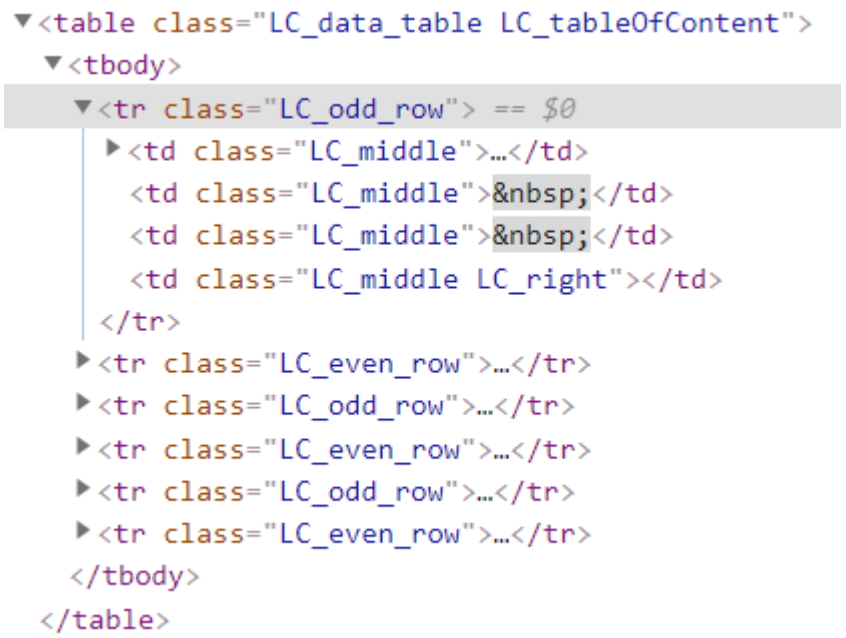


Abbildung 3.3: Lon-Capa Inhaltsverzeichnis HTML.

Der Aufbau dieser Tabelle ist sehr ähnlich zu dem des Kursauswahlfensters der Seite davor. In diesem Beispiel beinhaltet der Kurs 6 große Unterkapitel in Form von <tr>-Elementen. Jedes dieser Elemente enthält mehrere Unterelemente, wie beim ersten <tr>-Element erkennbar ist. Die 4 unteren Elemente sind Formatierungselemente, während das erste immer einen Link zu einer weiterführenden Seite in einem <a>-Element enthält. Dieser Link öffnet die gleiche Seite, nur mit dem jeweiligen Unterkapitel jeweils geöffnet bzw. geschlossen, wodurch die einzelnen Themen des Kapitels sichtbar werden. Auch diese werden in der Tabelle unter die Zeile des ausgewählten Unterkapitels angefügt und enthalten jeweils einen weiterführenden Link. Ein erneutes Klicken auf das ausgewählte Unterkapitel lässt diese Unterthemen wieder verschwinden.

```
> $$("[class^='LC_data_table'] td:first-of-type a:last-of-type")
< ▶ (6) [a, a, a, a, a, a]
```

Dieser Selektor selektiert alle <a>-Elemente innerhalb der Tabelle des Inhaltsverzeichnisses, welche die Links zu den weiterführenden Seiten enthalten. Da die Elemente für die Unterthemen bei dem Auswählen eines Unterkapitels einfach in die Tabelle eingefügt werden, kann dieser Selektor auch für diese veränderten Seiten wieder verwendet werden und gibt alle <a>-Elemente zurück.

```
> $$("[class^='LC_data_table'] td:first-of-type a:last-of-type")
< ▶(9) [a, a, a, a, a, a, a, a, a]
```

Folgt man einem der Links für die Unterkapitel auf die weiterführende Seite, erhält man, in diesem Beispiel, mit dem gleichen Selektor 3 neue Links zusätzlich zu den 6 vorher schon existierenden.

Um nun den Link zu den weiterführenden Seiten zu erhalten, muss nur jeweils das *href*-Attribut aller <a>-Elemente abgefragt werden.

Mit diesen Selektoren können die Grundfunktionen für die Navigation zwischen den Seiten abgedeckt werden. Es ist möglich, sich einzuloggen, Kurse auszuwählen und den jeweiligen Inhalt auszuwählen. Als nächstes werden Selektoren für die einzelnen Endknoten gesucht, also PDF-Dateien und den Seiten mit den Übungsaufgaben.

3.3.4 Aufgaben-Seite

Es existieren mehrere verschiedene Aufgabentypen, welche verschiedene Input-Methoden besitzen, um mit ihnen zu interagieren. Bei Aufgaben mit Eingabefeldern muss es für den Benutzer möglich sein, Antworten in Textform zu geben, und für Dropdown-Menüs oder Radio-Buttons müssen die verschiedenenen Antwortmöglichkeiten auswählbar sein. Der strukturelle Aufbau dieser Aufgaben ist allerdings grundsätzlich gleich. Alle notwendigen Elemente befinden sich immer innerhalb eines <form>-Elements.

```
{...}
    <form>
        Aufgabenbeschreibung

        <input> <!-- Antworttyp -->
        Antwortfeld 1
        <select></select> <!-- Antworttyp -->
        Antwort-Dropdown-Menü

        <table>
            <tr>
                <td>
                    <input> <!-- Formular abschicken -->
                    Antwort einreichen
                </td>
                <td>
                    <span></span> <!-- Anzahl Versuche -->
                    Versuche {...}/99
                </td>
            </tr>
        </table>
{...}
```

Abbildung 3.4: Lon-Capa AufgabenLayout HTML.

Dies ist der grundlegende Standardaufbau des <form>-Elements jeder Aufgabenseite. Je nach Aufgabentyp verändern sich die Antwortmöglichkeiten und die Aufgabenbeschreibung.

Um den Text der Aufgabenbeschreibung zu selektieren, müssen einige Dinge beachtet werden. Als erstes ist es nicht möglich, Freitext, um welchen es sich bei diesem Text handelt, mittels eines CSS-Selektors zu selektieren, da hierfür immer ein HTML-Element benötigt wird. Deswegen wird in diesem Fall XPath verwendet.

```
> $x("//body/form[@name='lonhomework']/input[@value='yes']/following-
  sibling::text()")
< ▶ (13) [text, text, text, text, text, text, text, text, text, text,
  text, text, text]
```

Dieser Selektor ermöglicht es, den gesamten Text des <form>-Elements zu selektieren. Da sich die Aufgabenbeschreibung immer am Anfang des Formulars befindet, ist sie üblicherweise in der ersten „text“-Rückgabe. Dies ist aber nicht immer der Fall,

da bestimmte mathematische Ausdrücke mittels der JavaScript-Library MathJax[2] dargestellt werden. Der Text, der sich in diesen MathJax <script>-Elementen befindet, wird dementsprechend nicht von diesem Selektor selektiert und muss separat abgefragt werden. Da in dieser Arbeit aber hauptsächlich untersucht wird, ob LON-CAPA überhaupt mittels Web scraping emulierbar ist, werden die MathJax-Elemente durch RegEx[3]-Ausdrücke durch normalen Text ersetzt und können so wie der andere Text auch mithilfe des Selektors selektiert werden. Allerdings werden auch damit nicht alle Textfelder abgedeckt, da sich *following-sibling* nur auf Elemente, welche auf gleicher Ebene des Ausgangselement sind, bezieht. Für diese Spezialfälle müssen manuell neue Testfälle geschrieben werden.

Dropdown-Menü

Als erster Aufgabentyp werden Dropdown-Menüs betrachtet. Zuerst muss für diese erfasst werden, wo sie sich auf der Seite befinden. Um mit Dropdown-Menüs interagieren zu können, muss bekannt sein, welche Optionsmöglichkeiten existieren. Als nächstes muss es für den Benutzer eine Möglichkeit geben, eine dieser Antwortoptionen auszuwählen.

```
▼<select onchange="javascript:setSubmittedPart('0');"
name="HWVAL_11:1">
    <option></option>
    <option value="rein reell">rein reell</option>
    <option value="echt komplex">echt komplex</option>
  </select>
```

Dies ist ein beispielhafter Aufbau eines Dropdownmenüs auf einer LON-CAPA Aufgabenseite. Die beiden Auswahlmöglichkeiten sind in diesem Fall „rein reell“ und „echt komplex“ werden durch die zwei <option>-Elemente repräsentiert.

```
> $$("[name='lonhomework'] select")
< ▶ [select]
> $$("[name='lonhomework'] select option")
< ▶ (3) [option, option, option]
```

Mithilfe dieser beiden Selektoren kann das ganze Dropdownmenü bzw. dessen Optionsmöglichkeiten selektiert werden. Bei jedem Menü existiert eine zusätzliche Optionsmöglichkeit, welche allerdings leer ist.

[2]MathJax - Library zum darstellen von mathematische Ausdrücken direkt im HTML-Code `https://www.mathjax.org/`
[3]RegEx - Regular expression (reguläre Ausdrücke

Radio-Button

Ein weiterer Aufgabentyp ist der Radio-Button. Es existieren immer zwei oder mehr Radio-Buttons, welche der Benutzer auswählen kann. Wird ein Button ausgewählt, nachdem ein anderer ausgewählt wurde, ist der Button, der zuerst ausgewählt wurde, nicht mehr ausgewählt (Es existieren auch Radio-Buttons, bei welchen mehrere Antworten ausgewählt werden können, diese werden allerdings nicht betrachtet). Nach dem empirischen Überprüfen mehrerer verschiedener Aufgaben, in denen Radio-Buttons benutzt werden, macht es den Anschein, als wäre der Aufbau dieser immer gleich. Alle Radiobuttons befinden sich als <input>-Element mit dem Typ *radio* in einem äußeren <label>-Element. Unterhalb des <input>-Elements befindet sich der zum jeweiligen Radio-Button gehörige Text.

```
> $x("//body/form[@name='lonhomework']/label/input/following
  -sibling::text()")
<- ▶(2) [text, text]
```

Allerdings müssen Radio-Button Elemente nicht zwangsläufig diesem Aufbau entsprechen, weswegen es vorkommen kann, dass dieser Selektor nicht immer funktioniert.

Eingabefelder

Aufgaben mit Eingabefeldern müssen vom Benutzer mit einfachen Tastatureingaben beantwortet werden. Hierbei muss beachtet werden, dass mathematische Ausdrücke korrekt eingegeben werden und der Nutzer informiert wird, falls seine Antwort ein dementsprechend falsches Format hatte. Diese Eingabefelder werden als <input>-Element mit dem Attribut *onkeydown* dargestellt. Mithilfe des Selektors

```
> $$("[name='lonhomework'] input[onkeydown]")
<- [
   ▶input#HWVAL_0_11_1_7_2.LC_textline.spellchecked.spellcheck
   -container
   ]
```

können diese Textfelder selektiert werden.

Weitere Funktionalitäten

Neben den Elementen zur Bearbeitung der Aufgaben existieren noch einige weitere Funktionen innerhalb der Aufgabenseite, welche für die Bearbeitung der Aufgaben nötig sind. Zum einen befindet sich der Button zum Abschicken des Formulars unterhalb der Aufgabe und zum anderen ein Textfeld, welches anzeigt, wie viele Versuche der Benutzer

für diese Aufgabe schon verbraucht hat. Diese beiden Elemente sind Unterelemente eines <table>-Elements und sind selektierbar mithilfe dieser Selektoren (1. Button, 2. „Versuche“-Text).

```
> $$("[name='lonhomework'] table tbody tr td input")
◂ ▶ [input#submit_0.LC_hwk_submit]
> $$("[name='lonhomework'] table tbody tr td span")
◂ ▶ [span.LC_nobreak]
```

Als letztes existiert noch ein weiteres Element, welches allerdings beim erstmaligen Öffnen einer jeden Aufgabenseite nicht sichtbar ist. Dieses Element wird erst nach dem Abschicken des Formulars sichtbar und zeigt an, ob die eingegebene Lösung für die Aufgabe korrekt, falsch oder ob ein Formatierungsfehler bei der Eingabe vorhanden war. Auch dieses Element befindet sich im <table>-Element der anderen Funktionalitäten.

```
> $$("[name='lonhomework'] table tbody tr td[class*='LC_answer']")
◂ ▶ [td..LC_answer_correct]
```

3.3.5 Seitenübergreifende Funktionen

Es existiert eine seitenübergreifende Funktion, die benötigt wird, um zwischen den einzelnen Funktionen zu wechseln. Diese besteht aus 2 Buttons, mit welchen eine Seite vorwärts und eine zurück navigiert werden kann.

```
> $$("#LC_breadcrumbs .LC_breadcrumb_tools_navigation
  [class='LC_menubuttons_link']")
◂ ▶ (2) [a.LC_menubuttons_link, a.LC_menubuttons_link]
```

Außerdem muss es noch möglich sein, von den Aufgabenseiten zurück zum Inhaltsverzeichnis des jeweiligen Kurses und der Kursauswahl zu gelangen. Um zum Inhaltsverzeichnis zu gelangen, existiert ein bestimmtes Element, welches bei Klick eine Javascript-Methode ausführt, die die Seite des Inhaltsverzeichnisses lädt. Die Kursauswahlseite kann geladen werden, indem der Link der Seite (der Anfangslink zu LON-CAPA) aufgerufen wird.

Dies sind alle benötigten Funktionen, um mit dem Clienten zu interagieren, Antworten abzugeben und das Formular abzuschicken.

4 Entwicklung des Programms

4.1 Anforderungsanalyse an das zu entwickelnde Programm

Es soll ein Programm entwickelt werden, mit dem es möglich ist, die Webseite LonCapa via Web Scraping zu steuern. Mit diesem soll es möglich sein, ein Großteil der Aufgaben zu bearbeiten.

4.1.1 Funktionale Anforderungen

Die wichtigste Funktion, die das entwickelte Programm bieten muss, ist die Möglichkeit, die Übungsaufgaben der Seite zu bearbeiten. Die wichtigsten und am häufigsten vorkommenden Aufgabentypen sollen dabei bearbeitbar sein. Diese drei sind:

- Dropdown-Menü
- Radio-Button
- Eingabefeld

Hierbei wird davon ausgegangen, dass sich der strukturelle Aufbau dieser Aufgabentypen zwischen verschiedenen Kursen nicht großartig unterscheidet.

Weitere Anforderungen an das System beinhalten:

- Die Serverantwort auf eine abgeschickte Aufgabe muss für den Benutzer einsehbar sein (korrekte Eingabe, falsche Eingabe, Fehler im Eingabeformat)
- Eine Ausgabe, falls das Format einer Aufgabe nicht bekannt ist und diese dementsprechend nicht anzeigbar ist
- Die Möglichkeit zur Navigation zwischen verschiedenen Aufgaben
- Auswahl der einzelnen Kurse auf der Kursauswahlseite
- Einloggen in das System

4.1.2 Nicht-funktionale Anforderungen

Um das Programm zu bedienen, muss eine Form der Eingabemöglichkeit für den Benutzer existieren. Um möglichst viele Aufgabentypen in der Entwicklung abzudecken, wird als Eingabemethode die Kommandozeile gewählt. Hierbei wird die Navigation zwischen den Aufgaben und auf der Inhaltsverzeichnisseite über Tastatureingaben erfolgen.

Der Quellcode, der für diese Navigation verantwortlich ist, wird dementsprechend extra markiert, und es wird darauf geachtet, dass diese Funktionalität weitestgehend gekapselt von dem eigentlichen Scrapingprogramm agiert, um einen möglichen späteren Austausch durch eine GUI-Oberfläche zu vereinfachen.

4.1.3 Technische Anforderungen

Getestet wurde das entwickelte Programm auf einem 64-Bit Windows 10 Rechner mit Python 3.8.0. Für die Ausführung des Programms müssen sowohl Python als auch die verschiedenen Frameworks auf dem Betriebssystem installiert sein, welche automatisch installiert werden können (siehe Anhang). Zusätzlich muss ein Browserdriver wie der Chromedriver auf dem System installiert sein. Für den Zugriff auf die Lon-Capa Website wird eine aktive Netzwerkverbindung benötigt.

4.2 Verwendete Technologien

Als Programmiersprache für die Entwicklung des Web Scrapers für Lon-Capa wurde Python gewählt. Grund dafür ist, dass sie zum einen eine sehr leicht zu lesende Sprache ist und zum anderen viele zusätzliche Libraries bietet, die den Umgang mit HTML-Webseiten vereinfachen. Als hauptsächliches Web Scraping Framework wird Selenium verwendet. Dies ist notwendig, da Lon-Capa oft mittels Javascript-Funktionen arbeitet, welche ohne einen Webdriver nicht ausgeführt werden können. Zusätzlich wird die Python Library `Requests` verwendet, welche notwendig ist, um den HTML-Quelltext einer Webseite herunterzuladen. Um mit regulären Ausdrücken zu arbeiten, welche für spezielle Aufgabentypen benötigt werden, wird das Python Modul `re` verwendet.

4.2.1 Selenium

Selenium wurde 2004 von Jason Huggins entwickelt und ist ein Open-Source Framework zur automatisierten Steuerung von Webseiten. Das Framework kann in vielen Programmiersprachen verwendet werden, unter anderem Java, C# und Python. Unter dem Begriff Selenium werden insgesamt 4 Tools zusammengefasst, welche alle für unterschiedliche Zwecke verwendet werden können [vgl. [Gur]].

Selenium IDE ist eine vollständige IDE, um mit Selenium Funktionen zu arbeiten und wird als Firefox Add-On und Chrome Extension implementiert. Dadurch ist es sehr einfach benutzbar, und es wird kein zusätzliches Wissen über andere Programmiersprachen benötigt. Aufgrund dieser Einfachheit können einige Funktionen wie Schleifen oder bedingte Operationen allerdings nicht durchgeführt werden, wodurch sich Selenium IDE nicht eignet, um komplexe Automatisierungen zu entwickeln.

Selenium Remote Control (Selenium RC) benutzt einen RC Server, um mit dem Browser zu kommunizieren, welcher als Zwischenelement zwischen diesem und den Seleniumbefehlen des Benutzers agiert. Dies tut der Server, indem er bei Start ein Javascript Programm in den Browser implementiert. Dieses empfängt dann Selenium Befehle direkt von dem Server, welche als Javascript Befehle ausgeführt werden. Diese Befehle steuern dann den Browser, wodurch Daten über den Server an den Benutzer, welcher die jeweiligen Befehle eingegeben hat, zurückgegeben werden. Selenium RC kann in verschiedenen Programmiersprachen implementiert werden und unterstützt auch Schleifen und bedingte Operationen. Durch den Server ist die Installation komplizierter als die von Selenium IDE, und es wird Wissen über mindestens eine weitere Programmiersprache benötigt, um mit Selenium RC zu arbeiten. Selenium RC ist außerdem offiziell veraltet und wurde von Selenium WebDriver abgelöst.

Selenium WebDriver ist das am meisten genutzte Tool des Selenium Frameworks. Anstatt wie bei Selenium RC über einen Server mit dem Browser zu kommunizieren, wird der Browser mittels des Selenium WebDrivers direkt vom System aus gesteuert. Dadurch wird kein zusätzliches Javascript benötigt, und es muss lediglich ein Webdriver auf dem System installiert sein. Dieser Webdriver öffnet bei Ausführung dann eine Browserinstanz, welche wie ein normaler Browser funktioniert und Seiten öffnen oder Buttonelemente klicken kann. Die gewünschten Informationen werden dann auf gleichem Weg über den Driver an den WebDriver zurückgegeben. Ohne den zusätzlichen Server wird so die Interaktion mit dem Browser schneller und die Performance des Programms wird verbessert. Wie beim Selenium RC wird allerdings zusätzliches Programmierwissen über einfaches HTML hinaus benötigt [vgl. [Bha]].

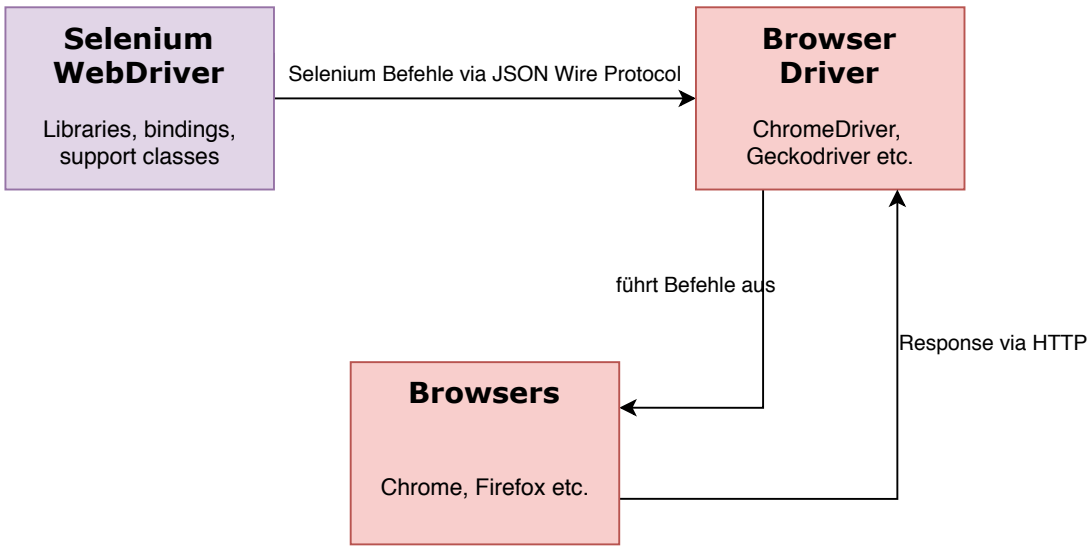


Abbildung 4.1: Selenium WebDriver Architektur.

4.2.2 Programmierung mit Selenium

An dieser Stelle werden kurz die Grundfunktionen von Selenium in programmatischer Ebene behandelt. Installiert werden kann das Framework über die offizielle Homepage[1]. Je nach Programmiersprache muss das Framework dann unterschiedlich in das Projekt eingebunden werden. Folgende Beispiele werden alle in der Programmiersprache Python dargestellt. Neben Selenium wird noch ein Browserdriver wie zum Beispiel der ChromeDriver oder Geckodriver benötigt, dessen Pfad nach Herunterladen in die Umgebungsvariablen des Systems hinzugefügt werden muss. Um nun Selenium innerhalb der gewünschten Programmiersprache zu verwenden, muss das Modul `webdriver` aus dem Modul `selenium` in das Projekt importiert werden. Als nächstes muss eine Instanz des jeweiligen Browserdrivers (in diesem Fall der ChromeDriver) mit dem Befehl

```
driver = webdriver.Chrome()
```

erstellt werden. Daraufhin öffnet sich ein Chrome-Browserfenster, auf welchem der Selenium Webdriver arbeitet. Um nun eine Webseite wie z.B. die LON-CAPA Homepage aufzurufen, kann die Methode

```
driver.get("http://loncapa.hs-hannover.de/")
```

verwendet werden. Selenium kann mit vielen verschiedenen Arten von Element-Selektoren verwendet werden, darunter auch CSS-Selektor und XPath. Um zum Beispiel alle <input>-

[1] Selenium Download - `https://www.selenium.dev/downloads/`

Elemente einer Seite mittels eines CSS-Selektors zu selektieren, kann folgender Ausdruck verwendet werden:

```
driver.find_elements_by_css_selector("input")
```

Alle `driver.find[...]` Methoden liefern als Rückgabe ein Objekt vom Typ Webelement bzw. eine Liste mit mehreren Objekten vom Typ Webelement. Diese Webelemente beinhalten verschiedene Methoden, um mit ihnen zu interagieren. Mithilfe der `getAttribute("attributname")` Methode können Attribute wie z.B. *href* des jeweiligen Elements ausgelesen werden. Desweiteren existieren Methoden wie `click()` und `send_keys("string_to_type")`, mit welchen das Element, falls es ein Button-Element ist, gedrückt werden kann und Zeichen in ein Textelement geschrieben werden können. Um mit einigen Elementen wie <select>-Elementen, interagieren zu können, müssen die entsprechenden Klassen aus dem Selenium-Modul zusätzlich importiert werden und das jeweilige Element auf diese Klasse gecastet werden. Zuletzt besitzen diese Webelement-Objekte wie der Webdriver auch `find[...]` Methoden, mit denen untergeordnete Elemente ab dem jeweiligen Startobjekt selektiert werden können.

Da der Driver über den Browserdriver permanent wie ein „normaler Nutzer“ mit den verschiedenen Webseiten interagiert, werden Sessions nach einem Einloggen auf einer Webseite von dem Server ganz normal angelegt und benutzt. Dadurch kann Selenium auch Daten auf Webseiten, die ein Login erfordern, sammeln. Selenium bietet auch die Möglichkeit, mithilfe der Methode `execute_script("javascript_script")` Javascript direkt auf der Browserinstanz auszuführen. Dies sind die wichtigsten Funktionen von Selenium, mit denen es möglich ist, die meisten Webseiten mittels Web Scraping zu bearbeiten.

4.3 Entwicklung des Scrapers

4.3.1 Funktionalität des Programms

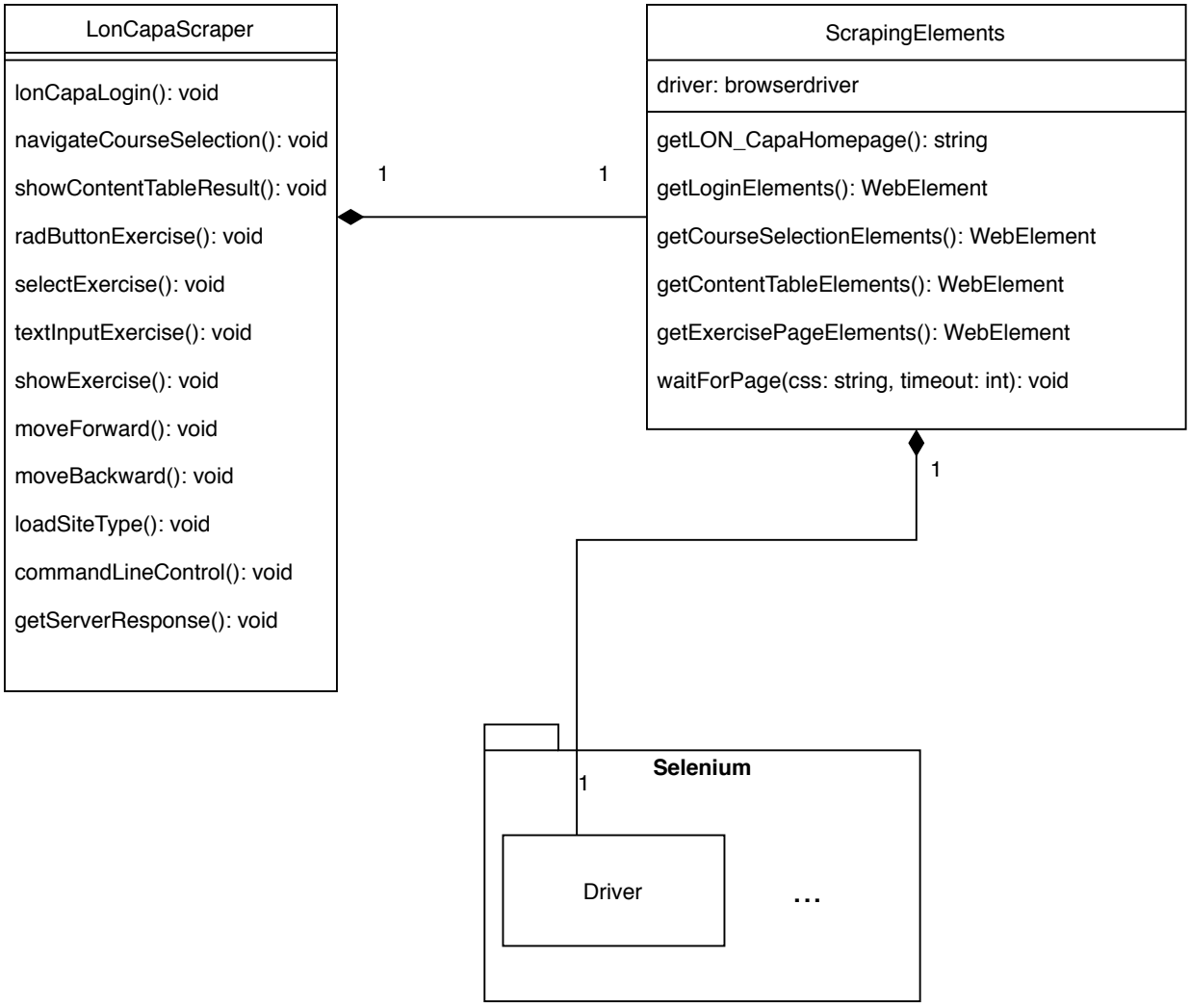


Abbildung 4.2: Klassendiagramm.

Die Hauptklasse des Programms ist der LonCapaScraper. Dieser enthält alle benötigten Funktionen, um mit der Lon-Capa Seite zu interagieren. Die Methoden `lonCapaLogin()`, `navigateCourseSelection()` und `showContentTableResult()` sind für die Funktionalität der Navigationsseiten von Lon-Capa zuständig. Ein Aufruf der Methoden lädt die entsprechenden Selektoren der Seite und gibt die gefundenen Elemente zurück, welche dann weiter verarbeitet werden. Für die Funktionalität der Aufgabenseiten existieren die Methoden `radButtonExercise()`, `selectExercise()` und `textInputExercise()`. Diese Methoden werden von der `showExercise()`, je nach vorhandenen Aufgabentyp auf-

gerufen. Werden auf der Seite beispielsweise <select>-Elemente gefunden, ruft diese die `selectExercise()`-Methode auf, welche die Elemente weiterverarbeitet. Nach der Ausführung einer dieser Funktionen für die jeweilige Aufgabenseite, wird die Methode `commandLineControl()` aufgerufen, welche auf die Eingabe des Nutzers wartet. Hier kann er auswählen, ob er die Aufgabe bearbeiten, zurück zur Inhaltsverzeichnisseite des Kurses, zur Kursauswahlseite selbst oder zur nächsten Aufgabe wechseln will.

Wählt der Nutzer nun aus, dass er die Aufgabe bearbeiten will, muss dieser je nach Anzahl der verschiedenen Unteraufgaben nach und nach die Lösung für die jeweilige Unteraufgabe eingeben. Am Ende wird der Benutzer noch gefragt, ob er die jeweilige Lösung zur Überprüfung durch den Server einreichen möchte. Stimmt er zu, wird das Formular samt den Eingaben des Nutzers an den Server geschickt. Durch die Methode `getServerResponse()` wird die Auswertung des Ergebnisses durch den Server abgefragt und ausgegeben, ob die Eingabe korrekt oder falsch war oder ob ein Formatierungsfehler bei der Eingabe vorlag oder nicht alle Unteraufgaben bearbeitet wurden. Um zwischen den einzelnen Aufgaben zu navigieren, existieren die Methoden `moveForward()` und `moveBackward`, welche über `commandLineControl()` aufgerufen werden.

Bei der Python Datei ScrapingElements handelt es sich um eine Klasse, welche die jeweiligen gewünschten Elemente zurückgibt und die Browserinstanz des Drivers hält, welcher durch Seleniummethoden gesteuert werden kann. Neben den in dem Klassendiagramm dargestellten Funktionen, die alle Elemente des Typs `WebElement`, zurückliefern existieren noch weitere strukturell gleiche Methoden für die anderen vorher definierten Selektoren. Zusätzlich enthält die Klasse noch eine weitere Funktion zum Warten auf das vollständige Laden der Lon-Capa Seite im Driver. Dies funktioniert, indem das Programm solange anhält, bis ein bestimmtes Element auf der jeweiligen Seite vorzufinden ist. Dies ist notwendig, da bei einer möglichen Verzögerung der Antwort des Lon-Capa Servers die Elemente nicht gefunden werden könnten.

Diese Aufteilung wurde gewählt, damit bei einer möglichen Änderung des Quelltexts der Lon-Capa Seite, die Selektoren in den betroffenen Methoden schnell und unabhängig voneinander geändert werden können.

4.3.2 Sequenzdiagramme

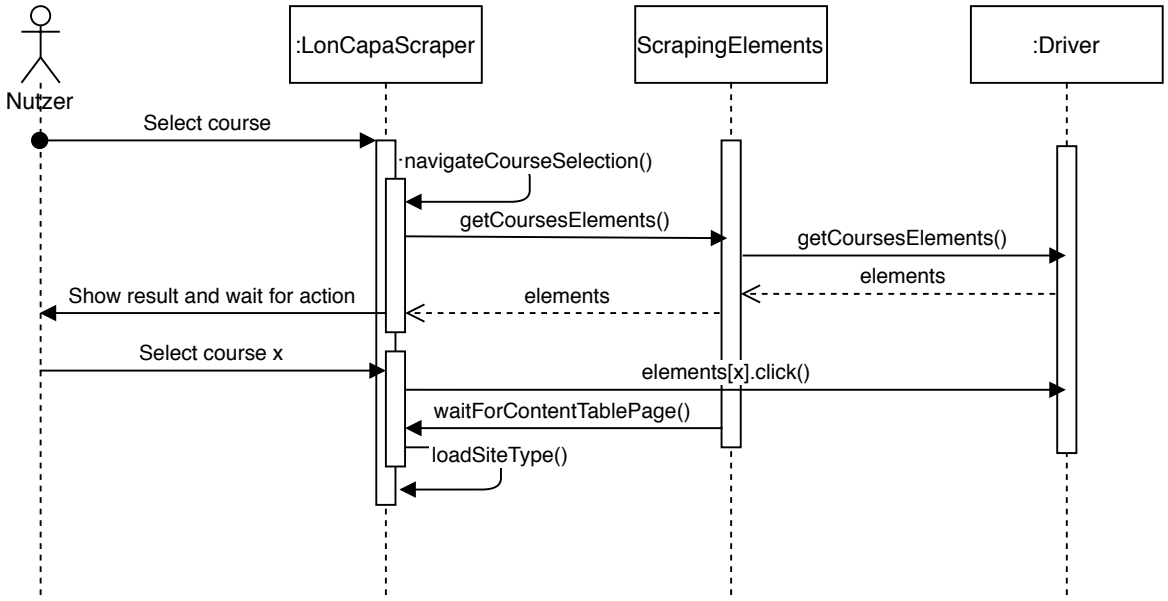


Abbildung 4.3: Sequenzdiagramm für den Anwendungsfall Kursauswahl.

Abbildung 4.3 zeigt einen Beispielablauf für die Interaktion des Benutzers mit der Kursauswahlseite. Jedes mal, wenn der Benutzer einen Kurs auswählen, möchte wird die Methode `navigateCourseSelection()` ausgeführt. Diese ruft die `getCoursesElements()` Methode des ScrapingElements-Moduls auf, welche die erforderlichen Elemente aus dem Driver ausliest und an den LonCapaScraper zurückgibt. Dieser gibt diese dann aus und wartet auf eine Eingabe des Benutzers. Je nach Auswahl des Benutzers wird das jeweilige Element ausgewählt, und der Driver bekommt die Aufforderung, das entsprechende Feld im Browser zu klicken. Danach wartet der LonCapaScraper mittels der `waitForContentTablePage()` Methode des ScrapingElements-Moduls, bis die Folgeseite fertig geladen ist. Zum Schluss wird noch die `loadSiteType()` Methode aufgerufen, welche nach einem Identifizierungselement auf der Folgeseite sucht und die zugehörige Methode des LonCapaScrapers, in diesem Fall `showContentResult()`, für das Inhaltsverzeichnis des Kurses aufruft.

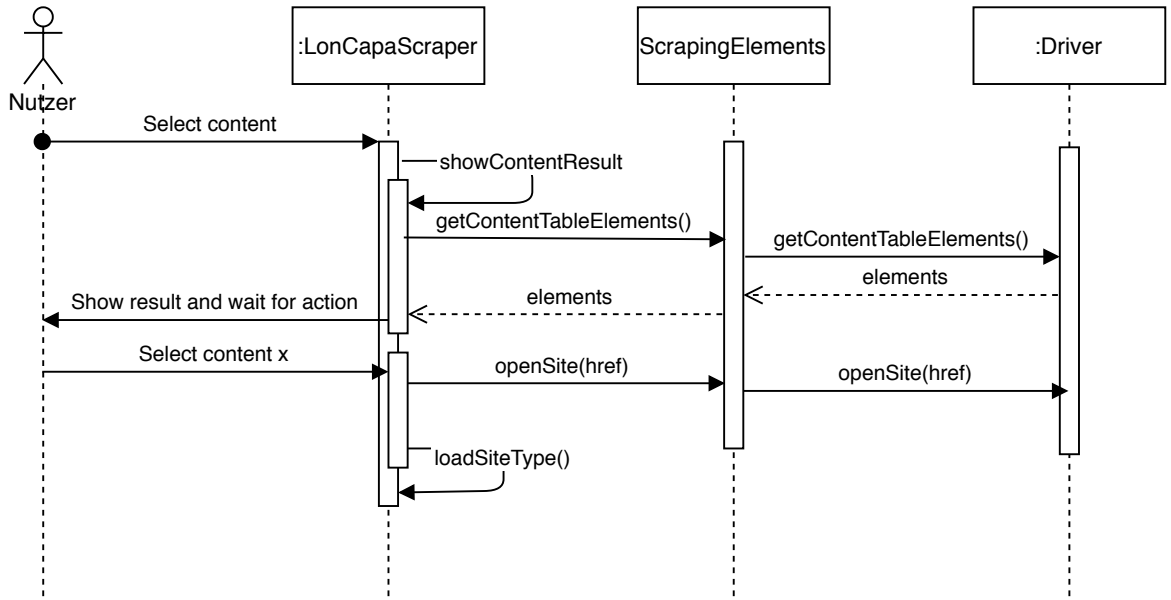


Abbildung 4.4: Sequenzdiagramm für den Anwendungsfall Inhaltsverzeichnis.

In *Abbildung 4.4* ist der Programmablauf bei Abruf der Inhaltsverzeichnisseite eines Kurses dargestellt. Auch dieser Programmteil beginnt damit, mittels der Methode `getContentTable()` die für die Seite entsprechenden Elemente zu selektieren und an den LonCapaScraper zurückzugeben. Danach wird auf eine Eingabe des Nutzers gewartet, dass er eines der Kapitel auswählt. Aus dem ausgewählten Kapitel, wobei es sich immer um ein <a>-Element handelt, wird dann das *href*-Attribut ausgelesen, welches die URL zu der entsprechenden Seite enthält. Durch die Methode `openSite(url)` des ScrapingElements-Moduls wird die Seite im Driver geöffnet. Nach vollständigem Laden der Seite ruft der LonCapaScraper `loadSiteType()` auf, um zu prüfen ob es sich bei der geöffneten Seite immer noch um eine Inhaltsverzeichnisseite oder eine Aufgabenseite handelt, und ruft die entsprechende Methode für diese Seiten auf.

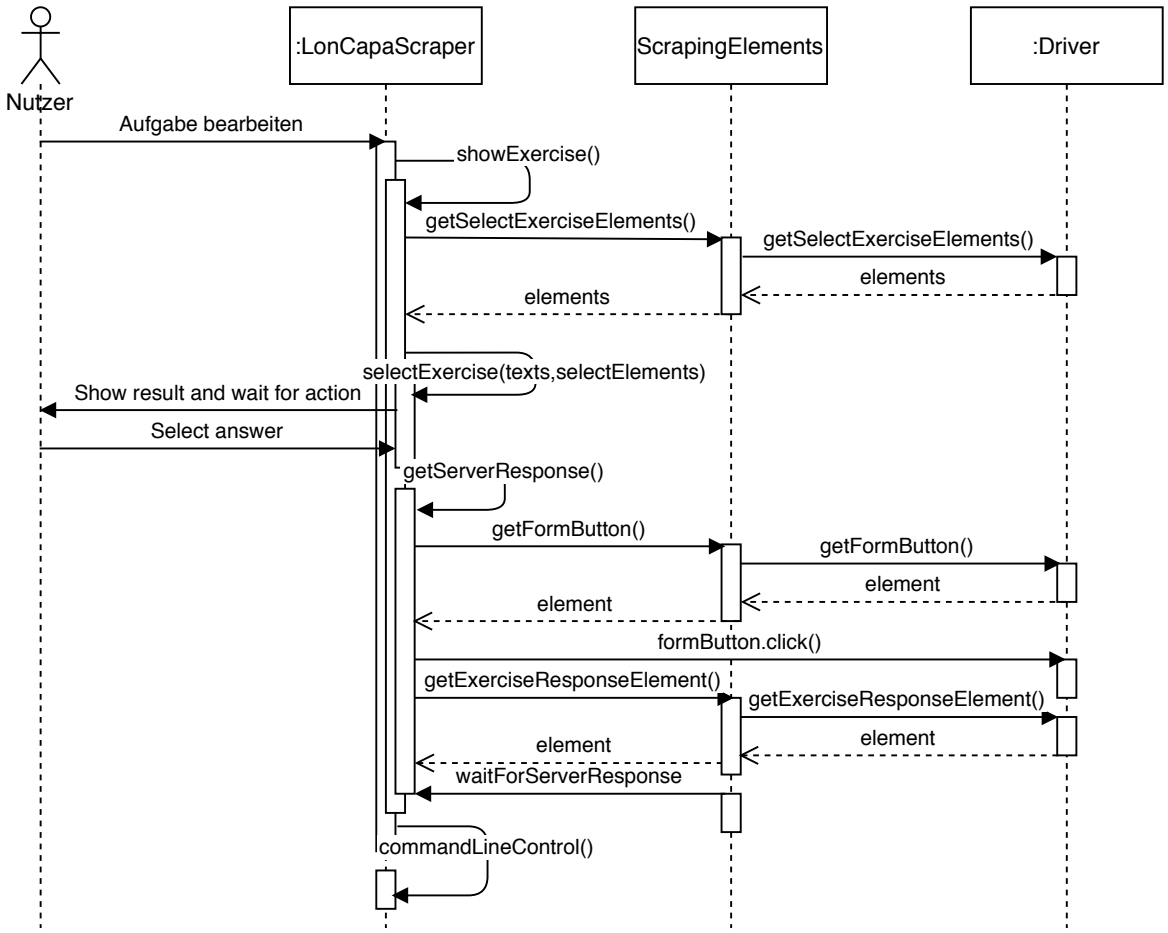


Abbildung 4.5: Sequenzdiagramm für den Anwendungsfall Aufgabenseite.

Abbildung 4.4 zeigt den Programmablauf bei Bearbeitung einer Aufgabenseite mit Dropdown Aufgaben. Am Anfang wird die `showExercise()` Methode aufgerufen, die über `getSelectExerciseElement()` die benötigten Elemente vom Driver an den LonCapaScraper zurückgibt. In diesem Beispiel handelt es sich um eine Seite mit dem Aufgabentyp Dropdown-Menü, weswegen `selectExercise(texts, selectElements)` mit den jeweiligen Elementen als Argumente aufgerufen wird. Nachdem der Nutzer eine der möglichen Optionen ausgewählt hat, wird die Methode `getServerResponse` aufgerufen, welche das Aufgabenformular abschickt und danach die Serverantwort mittels des zurückgegebenen Elements der `getExerciseResponseElement()` Methode ausgibt. Nach Ablauf des Programmteils wird `commandLineControl()` aufgerufen, was dem Nutzer die Möglichkeit zur nächsten Eingabe eröffnet.

4.4 Scraping durch manuellen Request

An dieser Stelle wird auf einen anderen Ansatz eingegangen, welcher es ermöglicht, ohne die Notwendigkeit einer Browserinstanz wie mit Selenium, mit dem Lon-Capa Server zu interagieren. Um dies zu erreichen, muss der *POST*-Request[2], der beim Abschicken des Formular-Elements an den Server geschickt wird, manuell verschickt werden.

4.4.1 Analyse des Requests

Dafür muss zuallererst der Aufbau des Requests analysiert werden, indem man ihn beim Abschicken aufzeichnet. Ein beispielhaftes Programm, welches so einen Request abfangen kann, ist Postman[3]. Neben dem Aufzeichnen von Requests, ermöglicht Postman zusätzlich noch das manuelle Verschicken eigener selbst erstellter *GET*-und *POST*-Requests.

Der HTTP Request-body mit der zugehörigen Aufgaben wird nachfolgend dargestellt.

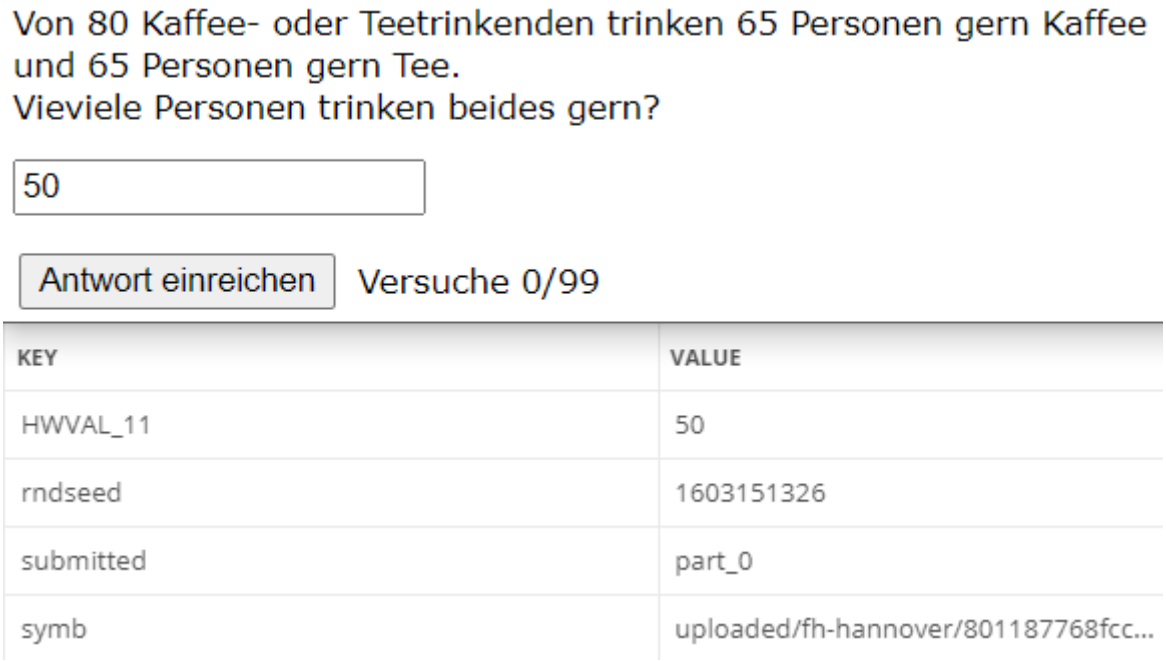


Abbildung 4.6: Lon-Capa Aufgabe mit Request-body.

[2]Post-Request - HTTP Request, welcher die Anfrage an den Server stellt, zugesendete Daten zu akzeptieren
[3]Postman - Plattform für API Entwicklung (`https://www.postman.com/`)

Der Key „HWVAL_11“ hat als Value den vom Benutzer eingegeben Wert. Existierten mehr als ein Eingabefeld auf der Seite, würden neben diesem noch weitere Keys mit ähnlichem Namen existieren. Dieser Key repräsentiert jeweils das Namensattribut des zugehörigen <input>-Elements. Neben den Keys für die Nutzereingabe existieren in dem Request-body noch weitere Einträge, wie ein Zufallswert mit dem Key „rndseed“, ein Key „submitted“ immer mit „part_0“ als Value und ein letzter Key „symb“ mit einer Zeichenkette als Wert.

KEY	VALUE
Upgrade-Insecure-Requests	1
Origin	http://loncapa.hs-hannover.de
Content-Type	multipart/form-data; boundary=----WebK...
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x6...
Accept	text/html,application/xhtml+xml,applicat...

Abbildung 4.7: Lon-Capa Aufgabe mit Request-header.

Im Request-header steht unter anderem, welche Art von Inhalt und von welchem User-Agenten dieser versendet wurde. Zusätzlich befinden sich noch Cookies[4], falls sie existieren, im HTTP Request-Header. Diese beinhalten in Lon-Capa eine ID, welche einen eingeloggten Nutzer identifiziert.

4.4.2 Request programmatisch erstellen

Um nun einen passenden Request programmatisch zu versenden, muss ein Request mit gleichem Body und Header, wie der Aufgezeichnete, erstellt und dann abgeschickt werden. Dafür werden die Python Bibliotheken *Requests* und *BeautifulSoup* verwendet. Requests ermöglicht es, wie der Name vermuten lässt, HTTP-Requests zu verschicken und den Quelltext einer Seite herunterzuladen. Zusätzlich kann auch eine Session erstellt werden, welche Session-Informationen wie Session Cookies speichern kann. BeautifulSoup kann den heruntergeladenen HTML-Code parsen und mittels verschiedenster Methoden, darunter auch CSS-Selektoren, nach bestimmten Elementen im Text suchen.

Bevor der POST-Request erstellt werden kann, muss zuerst eine Session mit dem Request Modul instanziiert werden. Dies ist notwendig, da die Aufgaben nur abgerufen werden

[4]Cookies - Daten welche zwischen Server und Client versendet werden, um Informationen über den Benutzer zu speichern

können, wenn der Nutzer eingeloggt ist. Der Session müssen dann die Cookies übergeben werden, welche den Benutzer identifizieren.

```
# cookies format
cookies = [{
    'domain': 'loncapa.hs-hannover.de',
    'httpOnly': True,
    'name': 'lonID',
    'path': '/',
    'secure': False,
    'value': 'loncapa_session_id' # dynamic session id
}]
```

Abbildung 4.8: Lon-Capa Identifizierungs-Cookie

Der „value“ muss hierbei die dynamisch generierte Lon-Capa Session-ID enthalten.

Nach vollständiger Instanziierung der Session kann mit dieser nun der Quelltext der Aufgabenseite durch Übergabe der jeweiligen URL heruntergeladen und geparst werden. Für Aufgaben mit Eingabefeldern kann der Selektor aus *Abschnitt 3.3.4 - Eingabefelder* verwendet werden, um alle <input>-Elemente zu finden und mithilfe von BeautifulSoup das Namensattribut der Elemente gefiltert werden.

```
# payload
payload = {
    'HWVAL_11': '50'
    'submitted': 'part_0'
}
```

Abbildung 4.9: Lon-Capa POST-Request Body

Der Body des POST-Requests setzt sich mindestens zusammen aus den Namen aller <input>-Elemente und den entsprechenden Eingaben des Nutzers sowie dem „submitted“ Key und dem zugehörigen Value aus *Abbildung 4.6*. Diese Daten werden mit der zugehörigen URL an die `post()` Methode der Sessioninstanz übergeben, welche nach Ausführung die Serverantwort zurückgibt. Diese Antwort kann nun wieder mit BeautifulSoup geparst und die Auswertung der Aufgabe ausgegeben werden.

Im Rahmen der Arbeit wurde solch ein Scraper mit limitierten Funktionen entwickelt. Initialisiert wird der Scraper mit der Lon-Capa Session-ID. Durch die Methode `showExercise(url)` scraped das Programm die Seite mit der entsprechenden URL

und fordert den Nutzer auf, Eingaben zu tätigen. Anschließend erstellt die Methode `createAndSendRequest(url, names, texts)` den POST-Request und schickt ihn an den Lon-Capa Server. Dieser Scraper funktioniert derzeit allerdings nur für Aufgabenseiten mit Eingabefeldern und einem Formular-Button.

4.4.3 Notwendige Voraussetzungen zur Nutzung dieser Methode

Das manuelle Verschicken des Requests benötigt einige Informationen um zu funktionieren. Zuallererst wird eine aktive Sitzung vorausgesetzt, dessen Session-ID für die Erstellung der Session aus dem Request-Modul notwendig ist. Ohne diese Session-ID leitet der Lon-Capa Server bei Aufruf der Aufgabe-URL sofort auf die Login-Seite weiter.

Eine weitere Voraussetzung ist, dass die URL der Aufgabe dem System bekannt sein muss. Diese ist grundsätzlich für jede Aufgabe ähnlich aufgebaut.

```
https://loncapa.hs-hannover.de/res/Domäne/Name/Modul/Kapitel/Aufgabe.problem
```

Grundsätzlich kann es auch vorkommen, dass in einem Kurs mehrere Aufgaben, in unterschiedlichen Ordnern, die gleiche URL für ihre Aufgaben haben. In diesem Fall wird noch der *symb*-Text aus *Abbildung 4.6* als Zusatz an die URL benötigt.

Damit geht einher, dass die Benutzerrollen innerhalb von Lon-Capa dafür sorgen, dass die Webseiten zu Aufgaben von anderen Kursen, trotz korrekter URL nicht aufgerufen werden können. Befindet man sich beispielsweise gerade im Kurs „Mathematik, WS 2020“ können Aufgaben des Kurses „Theoretische Informatik, WS 2020“ nicht aufgerufen werden. Dafür muss auf der Kursauswahlseite der entsprechende Kurs erst erneut ausgewählt werden.

4.4.4 Diskussion Scraping von Lon-Capa via manuellem Request

Um diese Methode sinnvoll und praktisch verwenden zu können, müsste der Zugang und die Darstellung der Aufgaben innerhalb von Lon-Capa verändert werden. Dieser Ansatz wäre praktisch umsetzbar, wenn auf die jeweiligen Aufgaben ohne Login zugegriffen werden könnte, denn so würde keine manuell erstellte Session benötigt werden, und die Seite könnte direkt bearbeitet werden.

Desweiteren müssten alle Aufgaben eindeutig und auch ohne Zuweisung einer Benutzerrolle zugänglich sein, damit sie frei bearbeitet werden können.

Ein solches Umstrukturieren des Servers würde allerdings dazu führen, dass der Lon-Capa Server keine Daten, wie die aktuellen Punkte mehr über die einzelnen Benutzer

speichern kann, da sie sich nicht bei ihm identifizieren. Diese Identifizierung müsste entweder durch Moodle passieren, oder die Punkte müssen direkt auf dem Moodle Server gespeichert werden.

5 Ergebnisse

5.1 Entwicklungsergebnisse

5.1.1 Funktionen und Grenzen des Web Scrapers

In diesem Kapitel wird überprüft, wie erfolgreich die Entwicklung des Web-Scrapers war und wo die Grenzen des Programms aktuell sind.

Im Allgemeinen lässt sich festhalten, dass der entwickelte Web-Scraper für LON-CAPA die funktionalen und nicht-funktionalen Anforderungen erfüllt. Er bietet die Möglichkeit, einen Großteil der drei aufgestellten Aufgabentypen zu bearbeiten und auswerten zu lassen. Weiterhin ist es möglich, frei zwischen den verschiedenen Aufgaben zu navigieren und Kurse auszuwählen. Auch das Einloggen in das LON-CAPA-System ist möglich.

Allerdings existieren auch einige Limitierungen in dem entwickelten Programm. Zum einen gibt es noch weitere Aufgabentypen in LON-CAPA, welche der Scraper nicht kennt und zum anderen kann es auch vorkommen, dass eine Aufgabe, trotz bekanntem Aufgabentyp, aufgrund von besonderen Formatierungen nicht korrekt dargestellt wird. Beispiele dafür sind Aufgaben mit Tabellen oder Graphen. Ein weiteres Problem ist, dass die gewählte Präsentationsmethode keine LaTeX-Ausdrücke interpretieren kann, weswegen Aufgaben, die in LaTeX dargestellt werden, einfach die LaTeX-Ausdrücke ausgeben.

5.1.2 Probleme bei der Entwicklung

Während der Entwicklung des Scrapers sind einige Probleme aufgetreten, die zum größten Teil allerdings gelöst werden konnten, wie z.B. das Warten auf das vollständige Laden einer Seite, bevor der Scraper diese verarbeitet. In Bezug dazu existiert allerdings noch ein Problem, welches im Rahmen der Arbeit nicht gelöst werden konnte. Elemente, die mithilfe von MathJax formatiert werden, werden nicht immer sofort im Quelltext angezeigt, wodurch es passieren kann, dass der LonCapaScraper diese nicht finden kann. Um dieses Problem zu lösen, müsste dieser bei jedem Aufruf einer Aufgabenseite explizit warten, bis ein MathJax-Element auf der Seite vorhanden ist. Da aber nicht auf jeder Aufgabenseite MathJax-Elemente verwendet werden, würde ein Warten auf diesen Seiten nie aufhören. Ein weiterer Ansatz wäre, auf jeder Seite grundsätzlich eine gewisse Zeit zu warten (ca. 1 Sekunde), was im Umkehrschluss allerdings die Ausführung des Programms verlangsamen würde.

5.1.3 Eignung für den Produktiveinsatz

Bezüglich eines möglichen Produktiveinsatzes des Scrapers muss betrachtet werden, wie das Programm auf einer Serverumgebung funktionieren kann. Da für jeden einzelnen Benutzer des Systems ein Webdriver mit einer eigenen Browserinstanz angelegt werden müsste, muss überprüft werden, wie der Server unter so einer Belastung noch arbeiten kann. Zudem muss eine geeignete Präsentationsoberfläche gewählt werden, welche die Eingabemethode über die Kommandozeile ersetzt.

Wenn anstatt des Selenium-basierten Web Scrapers ein Scraper verwendet werden soll, der wie in *Abschnitt 4.4* den Request manuell abschicken soll, muss hierfür die Funktionsweise des Lon-Capa Servers, wie in dem entsprechenden Abschnitt beschrieben, verändert werden.

5.1.4 Eignung von Web Scraping für LonCapa

An dieser Stelle wird abschließend diskutiert, ob sich Methoden des Web Scrapings nun für die Emulation von Lon-Capa eignen.

Die erfolgreiche Entwicklung des Scrapers zeigt, dass dies grundsätzlich möglich ist.

Allerdings bringt diese Methode einige Probleme mit sich. Zum einen verlangsamt sich die Ausführung des Systems im Vergleich zum direkten Bearbeiten der Aufgaben auf der LonCapa Website, da die Anzeigedaten nochmal zusätzlich heruntergeladen werden müssen. Unter normaler Benutzung der Seite bemerkt man diesen Unterschied allerdings kaum.

Ein größeres Problem ist, dass die Neuformatierung jedes Aufgabentyps viel weitere Arbeit an dem Programm benötigt. Eine mögliche Lösung für dieses Problem wäre, fast den gesamten Quelltext des <form>-Elements herunterzuladen und über eine HTML-Präsentationsschicht auf dem Zielsystem (Moodle) anzeigen zu lassen. Hierbei müssten dann noch dem <Button>-Element, welcher für das Abschicken des Requests zuständig ist, und den Eingabelementen die entsprechend neue Funktion gegeben werden. Durch eine Umsetzung dieser Vorgehensweise müssten zum einen nicht für jeden Aufgabentypen manuell Anwendungsfälle erstellt werden, und zum anderen würde die MathJax Integration automatisch funktionieren, da das jeweilige <script> immer mit heruntergeladen wird.

6 Fazit und Ausblick

In dieser Arbeit wurde untersucht, inwiefern sich die Website Lon-Capa mit Methoden des Web Scrapings emulieren lässt. Zu diesem Zweck wurde nach einer Einführung in das Thema ein Programm entwickelt, über welches die Website gesteuert werden kann.

Der Anfang der Arbeit bestand aus einem theoretischen Teil, in dem der Begriff „Web Scraping“ erläutert und einige Tools und Frameworks aus diesem Fachbereich vorgestellt wurden. Um das Programm zu entwickeln, wurden danach alle elementaren Funktionen der Lon-Capa Website beschrieben.

Anschließend wurden die vorgestellten Tools und Frameworks hinsichtlich ihrer Kompatibilität mit der Lon-Capa Website überprüft und bewertet.

Danach wurden die für diese Funktionen wichtigen Elemente der Webseite mithilfe von Selektoren herausgefiltert. Nach der Aufstellung einiger Anforderungen an das zu entwickelnde Programm wurde dies mit dem Selenium Web Driver Framework, welches es ermöglicht, mithilfe der aufgestellten Selektoren auf den Lon-Capa Clienten zuzugreifen, umgesetzt.

Neben dem Ansatz des Scrapings durch Selenium wurde auch noch ein anderer Ansatz beschrieben, durch den mithilfe des manuellen Abschickens des POST-Requests die Lon-Capa Aufgaben emuliert werden konnten.

Das im Rahmen der Arbeit entwickelte Programm erfüllt die aufgestellten Anforderungen.

Wie die Entwicklungsergebnisse aufzeigen, ist eine Emulation von Lon-Capa mit Web Scraping Methoden theoretisch möglich. Auch andere Funktionen, die im Rahmen der Arbeit nicht behandelt wurden, wie das Schreiben von Kommentaren oder das Anzeigen der erhaltenen Punkte, können mit dem Selenium Web Driver realisiert werden.

Wie schon in der Einleitung kurz erwähnt, war der Anreiz für dieses Thema, dass bestimmte Lon-Capa Funktionen innerhalb von Moodle verwendbar gemacht werden, damit der Lon-Capa Client nicht mehr direkt genutzt werden muss. In dieser Arbeit wurde untersucht, wie mit Lon-Capa durch ein Zwischenprogramm interagiert werden kann. Darauf aufbauend muss überprüft werden, wie das Programm nun in Moodle integriert werden könnte.

Eine Möglichkeit dafür wäre, das Programm auf einem Server laufen zu lassen. Auf diesem könnte dann durch eine Moodle Webseite zugegriffen werden. Hierbei müsste

aber, wie schon erwähnt, überprüft werden, wie der Selenium WebDriver serverseitig funktionieren kann.

Für den Request-Scraper aus Abschnitt 4.4 könnte beispielsweise eine Seite innerhalb von Moodle erstellt werden, welche die verschiedenen Aufgaben anzeigt. Beim Klick auf eine der Aufgaben würde das Programm mit der URL der Aufgabe ausgeführt werden und der Output auf der Seite ausgegeben werden. Hierfür müssten die Aufgaben allerdings ohne Login oder mittels der Moodle-ID zugänglich sein.

Literatur

[Bha] Debomita Bhattacharjee. *What is the Selenium Web Driver Architecture?* URL: https://www.tutorialspoint.com/what-is-the-selenium-web-driver-architecture#:~:text=Selenium%20WebDriver%20API%20enables%20interaction,Python%2C%20C%23%20and%20so%20on.. (Eingesehen am 20.10.2020).

[Gee] GeeksForGeeks. *Screen Scraping.* URL: https://www.geeksforgeeks.org/introduction-to-web-scraping//. (Eingesehen am 16.10.2020).

[Gur] Guru99. *What is Selenium? Introduction to Selenium Automation Testing.* URL: https://www.guru99.com/introduction-to-selenium.html. (Eingesehen am 20.10.2020).

[Rad] Radware. *An Overview of Web Scraping Techniques.* URL: https://www.shieldsquare.com/what-are-the-different-scraping-techniques/. (Eingesehen am 18.10.2020).

[Sel] Selenium. *Understanding the components.* URL: https://www.selenium.dev/documentation/en/webdriver/understanding_the_components/. (Eingesehen am 20.10.2020).

[Tec] Techopedia. *Legacy System.* URL: https://www.techopedia.com/definition/635/legacy-system. (Eingesehen am 17.10.2020).

[Tes] Testim. *XPath vs CSS Selector: The Difference and How to Choose.* URL: https://www.testim.io/blog/xpath-vs-css-selector-difference-choose/. (Eingesehen am 19.10.2020).

[MM] Daniel Myers und James W. McGuffee. *Choosing Scrapy.* https://www.researchgate.net/profile/James_Mcguffee2/publication/314179276_Choosing_Scrapy/links/58b8a088a6fdcc2d14d9a326/Choosing-Scrapy.pdf. (Eingesehen am 18.10.2020).

[S S] SCM de S Sirisuriya. *A Comparative Study on Web Scraping.* http://ir.kdu.ac.lk/bitstream/handle/345/1051/com-059.pdf. (Eingesehen am 17.10.2020).

[Nas] Mohamadou Nassourou. *Empirical Study on Screen Scraping Web Service Creation.* https://opus.bibliothek.uni-wuerzburg.de/opus4-wuerzburg/frontdoor/deliver/index/docId/4171/file/wrapper_webservice.pdf. (Eingesehen am 15.10.2020).

[Mit15] Ryan Mitchell. *Web Scraping with Python: Collecting Data from the Modern Web.* 2015.

7 Anhang

Installation unter Ubuntu

- Für die Ausführung müssen Python und die benutzten Module auf dem System installiert sein
- Installation von Python (`https://www.python.org/downloads/`)
- VirtualEnv installiereun und Projekt Environment erstellen (im Terminal ausführen):

```
pip3 install virtualenv
virtualenv scraperEnv
```

- Erstellte Environment aktivieren:

```
source scraperEnv/bin/activate
```

- Module installieren:

```
pip3 install -r requirements.txt
```

Starten des Programms

- Starten des Lon-Capa Scrapers:

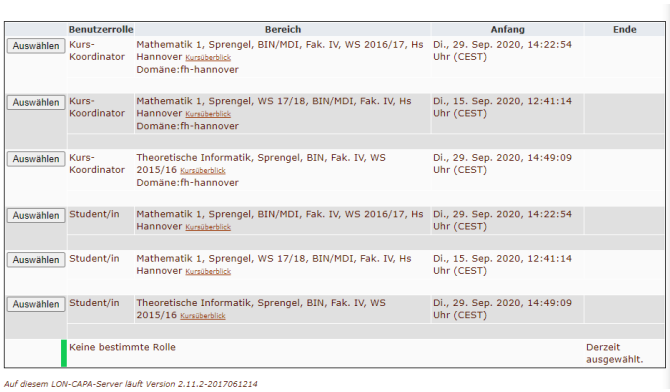
```
python .\SeleniumScraper\LonCapaScraperStarter.py
```

- Starten des Aufgaben Scrapers:

```
python .\RequestScraper\ExerciseScraperStarter.py <lonID> <URL>
```

LON-CAPA Webseiten

Kursauswahl



	Benutzerrolle	Bereich	Anfang	Ende
Auswählen	Kurs-Koordinator	Mathematik 1, Sprengel, BIN/MDI, Fak. IV, WS 2016/17, Hs Hannover Kursüberblick Domäne:fh-hannover	Di., 29. Sep. 2020, 14:22:54 Uhr (CEST)	
Auswählen	Kurs-Koordinator	Mathematik 1, Sprengel, WS 17/18, BIN/MDI, Fak. IV, Hs Hannover Kursüberblick Domäne:fh-hannover	Di., 15. Sep. 2020, 12:41:14 Uhr (CEST)	
Auswählen	Kurs-Koordinator	Theoretische Informatik, Sprengel, BIN, Fak. IV, WS 2015/16 Kursüberblick Domäne:fh-hannover	Di., 29. Sep. 2020, 14:49:09 Uhr (CEST)	
Auswählen	Student/in	Mathematik 1, Sprengel, BIN/MDI, Fak. IV, WS 2016/17, Hs Hannover Kursüberblick	Di., 29. Sep. 2020, 14:22:54 Uhr (CEST)	
Auswählen	Student/in	Mathematik 1, Sprengel, WS 17/18, BIN/MDI, Fak. IV, Hs Hannover Kursüberblick	Di., 15. Sep. 2020, 12:41:14 Uhr (CEST)	
Auswählen	Student/in	Theoretische Informatik, Sprengel, BIN, Fak. IV, WS 2015/16 Kursüberblick	Di., 29. Sep. 2020, 14:49:09 Uhr (CEST)	
	Keine bestimmte Rolle			Derzeit ausgewählt.

Auf diesem LON-CAPA-Server läuft Version 2.11.2-2017061214

```
1
Kurs-Koordinator
Mathematik 1, Sprengel, WS 17/18, BIN/MDI, Fak. IV, Hs Hannover Kursüberblick
Domäne:fh-hannover
Di., 15. Sep. 2020, 12:41:14 Uhr (CEST)

2
Kurs-Koordinator
Theoretische Informatik, Sprengel, BIN, Fak. IV, WS 2015/16 Kursüberblick
Domäne:fh-hannover
Di., 29. Sep. 2020, 14:49:09 Uhr (CEST)

3
Student/in
Mathematik 1, Sprengel, BIN/MDI, Fak. IV, WS 2016/17, Hs Hannover Kursüberblick
Di., 29. Sep. 2020, 14:22:54 Uhr (CEST)

4
Student/in
Mathematik 1, Sprengel, WS 17/18, BIN/MDI, Fak. IV, Hs Hannover Kursüberblick
Di., 15. Sep. 2020, 12:41:14 Uhr (CEST)

5
Student/in
Theoretische Informatik, Sprengel, BIN, Fak. IV, WS 2015/16 Kursüberblick
Di., 29. Sep. 2020, 14:49:09 Uhr (CEST)

Welcher Kurs?
```

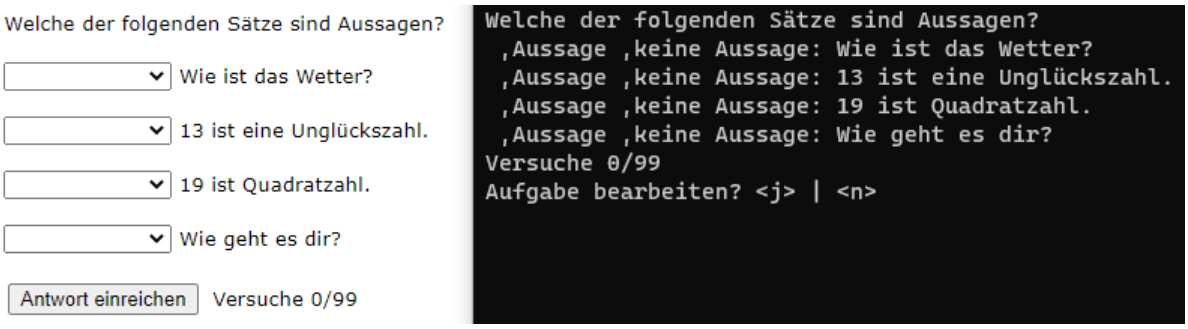
Inhaltsverzeichnis

- Kursüberblick
- Einführung
- Logik
- Mengen
- Ma0 - Potenzen
- Induktion
- Ma0 - Exponentialfkt. / Logarithmus
- Ma0 - Winkelfunktionen
- Komplexe Zahlen 1
- Komplexe Zahlen 2
- Ma0 - Ungleichungen und Betrag
- Relationen
- Relationen und Funktionen
- Funktionen
- Graphentheorie 1
- Graphentheorie 2
- Zahlentheorie 1
- Zahlentheorie 2
- Mathematik 1 - WS 2014/15 (Sprengel) - Videos
- Klausurvorbereitung
- Wiederholungsaufgaben
- Wiederholungsaufgaben ohne Punkte
- Aufgaben für Februar
- Gruppeneinteilung

```
0: Kursüberblick
1: Einführung
2: Logik
3: Mengen
4: Ma0 - Potenzen
5: Induktion
6: Ma0 - Exponentialfkt. / Logarithmus
7: Ma0 - Winkelfunktionen
8: Komplexe Zahlen 1
9: Komplexe Zahlen 2
10: Ma0 - Ungleichungen und Betrag
11: Relationen
12: Relationen und Funktionen
13: Funktionen
14: Graphentheorie 1
15: Graphentheorie 2
16: Zahlentheorie 1
17: Zahlentheorie 2
18: Mathematik 1 - WS 2014/15 (Sprengel) - Videos
19: Klausurvorbereitung
20: Wiederholungsaufgaben
21: Wiederholungsaufgaben ohne Punkte
22: Aufgaben für Februar
23: Gruppeneinteilung

Welcher Inhalt?
```

Aufgabenseite



Aufgabenseite Antwort