



Digitale Kompetenzen für Geistes- und Sozialwissenschaftler:innen. Vorzüge eines Blended Learning-Formats für die Vermittlung von Programmierkenntnissen.

Y. Frommherz*, J. Langenhorst

Professur für Angewandte Linguistik, Institut für Germanistik, Technische Universität Dresden

Abstract

Die Geistes- und Sozialwissenschaften (GSW) werden zunehmend digital. Wissenschaftler:innen stehen mittlerweile relevante Daten in großer Quantität zur Verfügung – aus dem Internet und durch die Digitalisierung analoger Bestände. Die neue Masse an Daten ermöglicht und erfordert den Einsatz quantitativer Analysemethoden. Zwar existieren niedrighschwellige Tools für einzelne Arbeitsschritte bei der Erschließung und Auswertung von Daten innerhalb der digitalen GSW, doch bieten diese lediglich vordefinierte Schnittstellen und schränken Wissenschaftler:innen somit ein. Grundlegende Programmierkenntnisse können hier Abhilfe schaffen, indem sie Forschenden ermöglichen, ihre Fragestellungen wesentlich flexibler zu verfolgen. Die Professur für Angewandte Linguistik bietet daher seit Langem einen Programmierkurs an, der sich spezifisch an den Bedarfen Linguistikstudierender ausrichtet und diesbezüglich stetig verbessert wird. Die Umstellung auf ein Blended Learning-Format im Video-Tutorial-Stil erwies sich als sehr geeignetes Setting: Studierende können die Inhalte räumlich-zeitlich flexibel konsumieren und so ihren Fortschritt individuell gestalten. Dieser Beitrag liefert einen ausführlichen Bericht über die Vermittlung von Programmierkenntnissen an Linguistikstudierende, diskutiert Vor- und Nachteile des umgesetzten Blended Learning-Formats und bietet einen Ausblick auf ein laufendes Projekt, in dem der bestehende Kurs optimiert und auf Studierende anderer GSW ausgeweitet wird.

The Humanities and Social Sciences (HSS) are subject to a digital turn. Scientists can now access relevant data in great quantities from the internet and by digitalizing analog data sources. This new bulk of data both enables and requires the use of quantitative methods. While low-threshold software does exist for specific steps within digital HSS research, these tools predefine how researchers can interact with their data and thus limit them. With basic programming skills, researchers can overcome this constraint and pursue their research goals much more flexibly. Therefore, the Chair of Applied Linguistics has long been offering a programming course. This course targets the specific needs of students of linguistics and has continuously been improved in this regard. Especially switching to a Blended Learning format, primarily using video tutorials, has proved to be an adequate setting: Students can study materials at their own pace wherever and whenever they wish. This article provides a detailed account of how programming skills can be taught to students of linguistics, discussing the (dis)advantages of the implemented Blended Learning format and giving an outlook on an ongoing project which aims at optimizing the course and extending it to students of other HSS.

*Corresponding author: yannick.frommherz@tu-dresden.de

1. Einleitung

Die Geistes- und Sozialwissenschaften (nachfolgend: GSW) sind in einem *digital turn* begriffen [1-3]. Es ist so einfach wie noch nie, große Mengen an geistes- und sozialwissenschaftlich relevanten Daten für die Forschung nutzbar zu machen – sei es aus dem Internet oder über die Digitalisierung analoger Datenbestände. Durch diese Entwicklung wird der Einsatz quantitativer Methoden gleichermaßen möglich und erforderlich [4, S. 297]. Der Schwerpunkt in den GSW liegt jedoch traditionell auf qualitativer Forschung. Neben den Techniken der Datenerschließung gehen diese neu relevanten quantitativen Methoden über die handwerklichen Fähigkeiten hinaus, die typischerweise in einem GSW-Studiengang vermittelt werden. Gleichzeitig sind aber auch viele niedrigschwellige Tools entstanden, mit denen diverse Arbeitsschritte innerhalb der digitalen werdenden GSW bewältigt werden können. So gibt es etwa einfach zu bedienende Software zur Untersuchung von Texten auf darin vorkommende Muster (etwa Schlüsselwörter), allerdings sind dabei die möglichen Analysekatoren immer bereits vordefiniert und somit eingeschränkt. Sobald Forschungsfragen verfolgt werden, für die diese Tools nicht entworfen wurden, sind zumindest grundlegende Programmierkenntnisse unerlässlich. Wenn sich Studierende und Forschende also einzig auf solche Software verlassen, können sie kaum das volle Potential ausschöpfen, das sich durch den *digital turn* ergibt.

Die Professur für Angewandte Linguistik an der TU Dresden vertritt daher die Auffassung, dass die Vermittlung von digitalen Kompetenzen im Allgemeinen und von Programmierkenntnissen im Besonderen ein notwendiger Bestandteil einer linguistischen Ausbildung ist. Um die Studierenden nachhaltig auf die zunehmend digitale Forschungs- und Arbeitswelt vorzubereiten, bietet die Professur seit vielen Jahren einen Programmierkurs an.

Eine Herausforderung in dieser Hinsicht ist, dass die Studierenden in aller Regel über keinerlei Vorkenntnisse im Programmieren und generell über wenig technische Kenntnisse verfügen. Deshalb stand bei der Ausgestaltung

des Programmierkurses von Anfang an die Frage im Zentrum, wie diese Fertigkeiten am besten an das gegebene Zielpublikum vermittelt werden können. Unter dieser Leitfrage wurde und wird der Kurs kontinuierlich weiterentwickelt und wurde so etwa von einem klassischen Präsenzseminar auf ein Blended Learning-Format umgestellt. In solch einem Setting werden Elemente der traditionellen, zeitlich synchronen und räumlich ko-präsenten Lehre mit online verfügbaren Lerninhalten verschiedener Medientypen kombiniert, die die Studierenden zeitlich asynchron und räumlich flexibel konsumieren können [5-7]. Dadurch können Studierende ihren Lernfortschritt weitgehend selbst steuern, was beim Erlernen des Programmierens Vorteile mit sich bringt (s.u.). Der vorliegende Beitrag liefert einen ausführlichen Erfahrungsbericht zur Vermittlung von Programmierkenntnissen an Studierende der Germanistik, und speziell der Angewandten Linguistik. Aufbauend auf einer detaillierten Argumentation, inwiefern Forschende der GSW von Programmierkenntnissen als fundamentale digitale Kompetenz profitieren (Kap. 2), wird die Ausgangslage, die zur Umstellung auf Blended Learning geführt hat, geschildert (Kap. 3). Kapitel 4 beschreibt den aktuellen Programmierkurs sowie die damit gemachten Erfahrungen im Blended Learning-Format. Kapitel 5 schließt mit einem Ausblick auf ein laufendes Lehrforschungsprojekt, das den bestehenden Kurs weiter optimiert und auf Studierende anderer GSW ausweitet. Der Beitrag richtet sich nicht zuletzt an interessierte Dozierende innerhalb der GSW, die ihre Studierenden ebenfalls auf die Möglichkeiten und Erfordernisse vorbereiten wollen, die durch den *digital turn* entstehen.

2. Warum brauchen Geistes- und Sozialwissenschaftler:innen Programmierkenntnisse?

Im Folgenden soll an konkreten Arbeitsschritten einer beispielhaften Studie demonstriert werden, weshalb Forschende in den GSW einen Nutzen aus Programmierkenntnissen ziehen. Wo sinnvoll, wird auf verwandte Techniken verwiesen, die ebenfalls grundlegender Programmierkenntnisse bedürfen, den For-

schenden im Gegenzug aber methodische Flexibilität ermöglichen. In der Beispielstudie soll mithilfe eines quantitativen Ansatzes untersucht werden, wie sog. Erfolgscoaches auf YouTube ihr Publikum sprachlich adressieren. Dabei handelt es sich um ein Thema, das im Rahmen einer studentischen Arbeit im Studiengang Linguistik untersucht werden könnte. Typischerweise geschähe dies auf qualitative Weise, d.h. der Fokus läge auf einzelnen Belegen (einzelnen Instanzen von *Publikumsansprache*), die einer eingehenden Analyse unterzogen würden.

Beim hier verfolgten quantitativen Ansatz soll jedoch eine große Menge an Daten maschinell ausgewertet werden, um so Muster erkennen zu können, die über einzelne Belege hinausgehen. Im Idealfall treten dadurch Regelmäßigkeiten zu Tage, die bei einer qualitativen Herangehensweise verborgen blieben. In jedem Fall lassen sich die Muster in den Daten aber *quantifizieren*, wodurch klare Aussagen zur Relevanz eines Phänomens im gegebenen Kontext getroffen werden können.

```
for id in video_ids:
    result = {}
    transcript = YouTubeTranscriptApi.get_transcript(id, languages=['de'])
    text = ""
    lines = []

    for line in transcript:
        lines.append(line['text'])

    result['title'] = title.replace(" - YouTube", "")
    result['video_id'] = id
    result['transcript'] = lines
```

Abb. 1: Code zum Herunterladen von YouTube-Transkripten (Ausschnitt).

```
{
  "title": "ERFOLG BASICS: Durch diese Regeln wurde ich Milliardär",
  "video_id": "waavw4CRtoo",
  "transcript": [
    "liebe youtube freunde es geht heute um",
    "erfolg weltlichen erfolg und es gibt nur",
    "einen glasklaren grund warum ich",
    "überhaupt befugt bin und fähig bin",
    "darüber zu reden",
  ]
}
```

Abb. 2: Gespeicherte YouTube-Transkripte.

In einem ersten Schritt müssen die Daten, also die in den Videos vorkommende Sprache, erschlossen werden. Mithilfe bestehender Module (d.h. grob formuliert, fertigen Codebausteinen für einen bestimmten Zweck) für die hier verwendete Programmiersprache Python sowie weniger selbst geschriebener Zeilen Code kann über die Video-IDs (jedes YouTube-Video verfügt über eine solche ID) auf die jeweiligen Untertitel zugegriffen werden. Diese

können dann inklusive relevanter Metadaten in einem geeigneten Format abgespeichert werden (Abb. 1 und Abb. 2).

Ähnliche Module, die es Forschenden erlauben, durch wenige Zeilen Code bestimmte Inhalte von Sozialen Medien herunterzuladen, gibt es für diverse Plattformen. Abseits davon können Forschende auch selbst *scrapen*, also gezielt Inhalte von beinahe allen Webseiten herunterladen (*Web Scraping*). Eine weitere Methode zur Datenerschließung, die Forschenden mit Programmierkenntnissen zur Verfügung steht, ist die Einbindung von Modulen zur *Optical Character Recognition (OCR)* in den Code. Diese macht es möglich, Schriftzeichen aus Bildern auszulesen, wodurch der extrahierte Text maschinell auswertbar wird, was sowohl bei historischen Quellen als auch bei sog. *Sharepics* (Bildern mit Text zum Teilen, etwa bei Instagram) interessant ist.

Der zweite Schritt besteht darin, die gesammelten Daten aufzubereiten. Unter Einsatz geeigneter Module können sie flexibel und effizient verwaltet, gefiltert und für die Auswertung vorbereitet werden. Für die Auswertung im Rahmen der Beispielstudie sollen die gesammelten Daten Wort für Wort vorliegen (s.u.), daher werden sie in diesem Schritt tokenisiert (Abb. 3).

In
Ihren
Videos
greifen
Sie
doch
mehrere
Themen
an
,
denken
Sie
nicht
,
dass
Sie
damit
jemanden
beleidigen
könnten
?

Abb. 3: Tokenisierter Text.

Im dritten Schritt werden die Daten ausgewertet. Dafür wird *Publikumsansprache* als Kollokationen der Personalpronomina *Du* und *Sie* operationalisiert. Kollokationen sind häufig zusammen auftretende Wörter, etwa *unbeschrieben* und *Blatt* in der allgemeinen Sprache. Hier soll nun analysiert werden, welche Wörter häufiger als aufgrund ihrer Häufigkeit im sonstigen Text erwartbar in der Umgebung der ansprechenden Pronomina auftreten (Abb 4. zeigt dies für *Du*, $N = 4$ Videos).

collocate	log Dice
kannst	12.23
hast	12.07
liebst	11.78
bist	11.76
wenn	11.46
etwas	11.4
lernen	11.14
tust	11.07
musst	10.84
was	10.82
findest	10.62
wirst	10.58

Abb. 4: Kollokationen zu ‚Du‘.

Für *Du* zeigt sich in Abb. 4 unter anderem, dass das Modalverb *können* am häufigsten (im Vergleich zur erwartbaren Häufigkeit) in den analysierten YouTube-Videos von Erfolgsscoaches vorkommt. Es scheint, als läge der von den untersuchten Coaches beworbene, sprichwörtliche Schlüssel zum Erfolg im intrinsischen Vermögen des Zielpublikums. Diese These kann nun zum Ausgangspunkt weiterer quantitativer und qualitativer Untersuchungen genommen werden.

Zusammengefasst können Forschende – wie an diesem Anwendungsfall demonstriert – nur mit Programmierkenntnissen relevante Daten auf effiziente Weise erschließen und diese dann methodisch flexibel, auf die jeweilige Fragestellung abgestimmt aufbereiten und auswerten. Ein weiterer Anwendungsbereich von Programmierkenntnissen ist zudem die Visualisierung von Daten. Wenngleich es zu diesem Zweck Software gibt, für die keine Programmierkenntnisse benötigt werden (z.B. SPSS oder Microsoft Office-Software), ermöglichen

Programmierkenntnisse eine wesentlich flexiblere Visualisierung, die gerade zum Zweck wissenschaftlicher Publikationen vorteilhaft ist.

Zusätzlich zu diesen konkreten methodischen Vorteilen von Programmierkenntnissen wird Forschenden durch die digitale Arbeit ermöglicht, sich *algorithmisches Denken* anzueignen bzw. dieses zu festigen [4, S. 95]. Unter algorithmischem Denken wird die Fähigkeit verstanden, 1) ein gegebenes Problem genau zu analysieren, 2) einfache Schritte zu seiner Lösung zu definieren, 3) aus diesen Schritten eine Lösung zu erstellen, d.h. einen vollständigen und korrekten Algorithmus zu konstruieren, 4) diesen Algorithmus auf alle typischen wie untypischen Fälle hin zu testen, und 5) seine Effizienz zu verbessern. Algorithmisches Denken ist Voraussetzung für Programmierung und erwächst gleichermaßen aus dem Umgang mit Code. Von der Übung im algorithmischen Denken profitieren Forschende auch abseits des Programmierens, etwa in ihrer wissenschaftlichen Arbeit, zumal eine gründliche, durchdachte Herangehensweise an ein Problem (1-3 oben) sowie die stete Bemühung, eine potenzielle Lösung zu testen (4) in beiden Fällen elementare Aspekte sind.

3. Ausgangslage

Da Programmierkenntnisse also eine zunehmend relevante digitale Kompetenz für Studierende darstellen, bietet die Professur für Angewandte Linguistik seit Langem den Kurs *Programmieren für Sprachwissenschaftler:innen* an. Vor der Umstellung auf Blended Learning wurde dieser wie bereits erwähnt als klassisches Präsenzseminar durchgeführt. Unterrichtet wurde die Skriptsprache Perl. Von Woche zu Woche wurden neue Inhalte im Seminar vermittelt und wöchentlich waren Aufgaben durch die Studierenden zu lösen. Ergänzend wurde zudem ein Tutorium angeboten, was für Seminare in der Linguistik sonst unüblich ist, sich aufgrund der inhaltlichen Dichte aber als unbedingt erforderlich erwies. Einzelne Seminarsitzungen wurden genutzt, um gemeinsam an Aufgaben zu arbeiten, während andere jeweils der Einführung in ein Konzept der Programmierung bzw. später im Semester in spezifisch sprachwissenschaftliche Techniken wie

Web Scraping und insbesondere Korpusauswertung dienen.

Über die Semester gelang es so, stets eine gewisse Zahl von Studierenden für das Programmieren zu begeistern und ihnen so neue Möglichkeiten für ihre eigenen Studien zu eröffnen. Allerdings führten gleichzeitig auch viele Teilnehmer:innen das Seminar nicht zu Ende – die Abbruchquote war im Vergleich zu anderen Seminaren an der Professur deutlich höher. In einem einzigen Semester Studierenden, welche größtenteils nie zuvor programmiert hatten und auch sonst oft wenig technische Vorkenntnisse besaßen, eine Vielzahl von Programmier-techniken zu vermitteln, stellte in diesem Modus eine Herausforderung dar.

4. Umstellung auf Blended Learning

Mit dem Wintersemester 2020/21 wurde der Kurs im Zuge der coronabedingten Umstellung auf digitale Lehre neu in einem Blended Learning-Format durchgeführt. Es wurden einerseits asynchrone Videoinhalte angeboten, andererseits wurde das Tutorium weiterhin synchron und – soweit es das Pandemiegeschehen zuließ auch räumlich ko-präsent – abgehalten. Lehrmaterialien verschiedener Medientypen, allen voran die genannten Videos, aber auch Code-Snippets, veranschaulichende Diagramme, ergänzende Dokumente sowie Links zu weiterführenden Ressourcen wurden miteinander verknüpft (Abb. 5 und Abb 6).



The screenshot shows a light blue page with the title 'Programmieren für Sprachwissenschaftler:innen' in a large, bold, black serif font. Below the title are three blue hyperlinks: 'Mail / Zu OPAL / Matrix-Raum', 'Python-Dokumentation', and 'Übersicht:'. The 'Übersicht:' section contains a numbered list of 12 sessions, each with a date and a list of topics in blue hyperlinks. At the bottom of the page is another blue hyperlink: 'Informationen zu den Prüfungsleistungen'.

Programmieren für Sprachwissenschaftler:innen

[Mail / Zu OPAL / Matrix-Raum](#)

[Python-Dokumentation](#)

Übersicht:

- [Sitzung 1 \(27.10.2020\) - Einrichtung](#)
- [Sitzung 2 \(03.11.2020\) - Kommentare, Variablen, User Input](#)
- [Sitzung 3 \(10.11.2020\) - Listen, for-Loops](#)
- [Sitzung 4 \(17.11.2020\) - Bedingte Anweisungen, Booleans](#)
- [Sitzung 5 \(24.11.2020\) - Kommandozeilen-Argumente, File Input/Output](#)
- [Sitzung 6 \(01.12.2020\) - Dictionaries](#)
- [Sitzung 7 \(08.12.2020\) - Unser erstes Korpus](#)
- [Sitzung 8 \(15.12.2020\) - Encodings, Funktionen](#)
- [Sitzung 9 \(05.01.2021\) - XML, N-Gramme](#)
- [Sitzung 10 \(12.01.2021\) - Scraping](#)
- [Sitzung 11 \(19.01.2021\) - XML schreiben, Stanza](#)
- [Sitzung 12 \(26.01.2021\) - Reguläre Ausdrücke](#)

[Informationen zu den Prüfungsleistungen](#)

Abb. 5: Überblicksseite des Kurses (Wintersemester 2020/21).

Die Kommunikation im Seminar erfolgte über einen Chatraum und weitere Beratung bei Problemen in erster Linie per Mail. Später im Semester wurden sog. *Echtzeit-Codings* als Video zur Verfügung gestellt, also Aufzeichnungen, in denen vollständige Programmieraufgaben in Echtzeit gelöst wurden, etwa das Scraping einer Webseite. Dabei wurden die einzelnen Schritte – Analyse der Seite, Links finden,

Seiten herunterladen, Text und Metadaten extrahieren und annotieren, in Korpus-Datei zusammenfassen – Stück für Stück vorgeführt, statt sie nur zu erläutern (Abb. 7). Diese Echtzeit-Coding-Sitzungen waren inspiriert von Coding-Streams auf der Streamingplattform Twitch bzw. auf YouTube, in denen man Entwickler:innen live – bzw. im Nachhinein bei der Aufzeichnung des Streams – beim Program-

mieren zuschauen kann. Solche Streams erfreuen sich großer Beliebtheit und stellen mittlerweile eine Art eigenes Lehrangebot dar [9][10]. Ein Unterschied zwischen dem im Se-

minar eingesetzten Echtzeit-Coding und den Twitch-Streams ist natürlich, dass bei aufgenommenen Videos nie Live-Interaktion mit den Zuschauer:innen stattfindet.

Sitzung 7: Dictionaries

Existiert ein Schlüssel?

```
capital_cities["Spanien"]  
→ Fehler wenn Schlüssel nicht existiert
```

```
capital_cities.get("Spanien")  
→ Entweder Wert oder None
```

```
capital_cities.get("Spanien", "unbekanntes Land")  
→ Entweder Wert oder default-Wert
```



TECHNISCHE UNIVERSITÄT DRESDEN
Programmieren für Sprachwissenschaftler:innen
Professur für Angewandte Linguistik / Jan Langenhorst
30.11.2021
Folie 7

[Folien \(PDF\)](#)

Dictionary deklarieren:

```
ages = {"Fridolin" : 27, "Hannah" : 22}
```

Abb. 6: Kursinhalte.

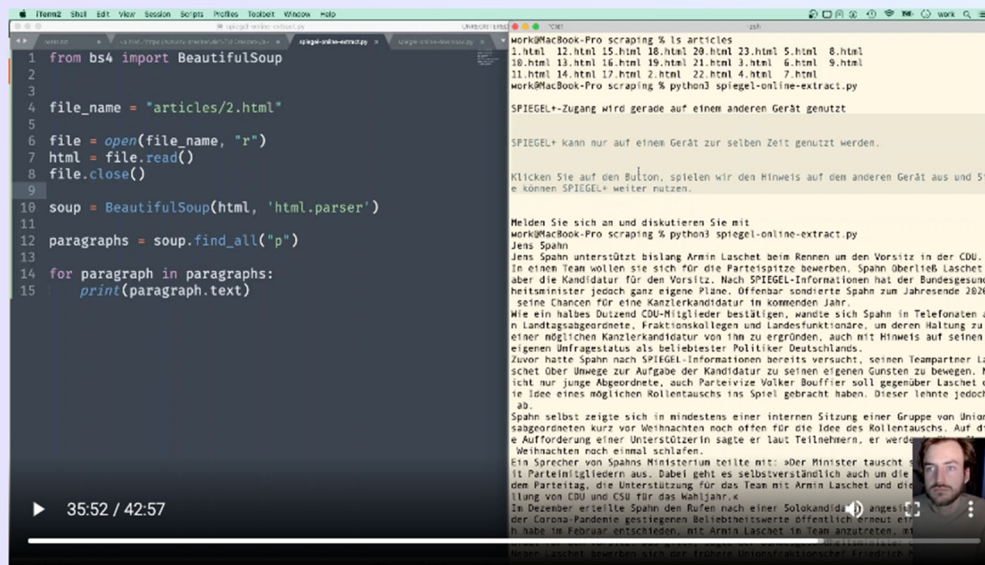
Abgesehen vom neuen Format war eine weitere Neuerung, dass Python statt wie bisher Perl unterrichtet wurde, da sich Python mittlerweile zu einer Art Wissenschaftsstandard entwickelt hatte. Python bietet zahlreiche Bibliotheken zur Analyse strukturierter Daten (u.a. Pandas), Datenvisualisierung (u.a. Matplotlib), Annotation (u.a. die NLP-Bibliotheken Stanza, SpaCy, NLTK, TextBlob etc.) sowie zum Scraping (u.a. BeautifulSoup, Selenium). Letztlich ist Python aufgrund seiner etwas simpleren Syntax und Semantik möglicherweise für Anfänger:innen leichter zu erlernen als das zuvor unterrichtete Perl (zum Vergleich von Python/Java: [11]).

Auch wenn vorhandene Bibliotheken einen Grund darstellen, eine spezifische Sprache zu unterrichten, so lag der Fokus des Seminars – als ein *Programmierseminar* – darauf, dass Studierende möglichst viel Code selbst entwi-

ckeln. Die Studierenden sollten nicht einfach vorhandenen Code mittels Python-Module wie in einem Baukasten zusammensetzen, sondern sich möglichst viele Schritte selbst erarbeiten. Durch reine Einführungen in Bibliotheken wäre kaum ein Mehrwert im Vergleich zu den eingangs erwähnten niedrigschwelligen Tools entstanden, die es auch für die Auswertung von (korpus-)linguistischen Daten gibt (z.B. Corpus Workbench, Sketch Engine). Es wäre z.B. möglich, sich von einem Python-Package innerhalb weniger Zeilen häufige Wortverbindungen ausgeben zu lassen. Mit etwas eigenem Aufwand kann dieser Prozess aber auch selbst nachvollzogen werden, wodurch die eigentliche Verarbeitung der Daten transparent wird. Somit steht man nicht einer Black Box in Form eines Tools gegenüber, sondern versteht die Analyse besser, da man die einzelnen Schritte selbst entwickelt hat.

Sitzung 10: Scraping

Scraping-Beispiel



Skript 1

```

import urllib.request
from bs4 import BeautifulSoup

urls = set()

# Folgende URLs wollen wir ausnehmen, obwohl sie mit /politik/ anfangen
exclude_urls = [
    "https://www.spiegel.de/politik/",
    "https://www.spiegel.de/politik/deutschland/",
    "https://www.spiegel.de/politik/ausland/",
    "https://www.spiegel.de/politik/deutschland/",
    "https://www.spiegel.de/politik/ausland/",
    "https://www.spiegel.de/politik/ausland/",
    "https://www.spiegel.de/politik/"
]

# Übersichtsseite laden
response = urllib.request.urlopen("https://www.spiegel.de/politik/deutschland/")
data = response.read()
soup = BeautifulSoup(data, 'html.parser')

# Alle a-Elemente (= Links finden, egal wo sie stehen)
links = soup.find_all("a")

for link in links:

```

Abb. 7: Echtzeit-Coding-Video mit zugehörigem Code-Snippet.

Nachdem die Einrichtung von Python gemeistert war und sich die Studierenden mit der Shell vertraut gemacht hatten, wurde inhaltlich bei den Grundlagen der Programmierung wie Kontrollstrukturen und Datentypen sowie zugehörigen simplen Übungen angefangen. Ab Mitte des Semesters wurden immer spezifischere korpuslinguistische Anwendungen erarbeitet. Die Korpuslinguistik untersucht sprachliche Phänomene auf Basis großer Textsammlungen (*Korpora*), welche linguistisch (z.B. mit Wortarten) und mit Metadaten (z.B. Autor:in, Jahr, Genre) annotiert sind. Den Studierenden wurden zunächst unterschiedlich strukturierte Korpora zur Verfügung gestellt, welche auf häufige Wörter, n-Gramme (*n* aufeinanderfolgende Wörter) usw. untersucht werden sollten. Danach lag der Fokus auf der

Erstellung und Annotation eigener Korpora aus Webquellen. Insgesamt lernten die Studierenden Skripte zu schreiben, um selbst erstellte Korpora auf verschiedene sprachliche Muster sowie die Faktoren, die deren Auftreten bedingen, zu untersuchen. Aus den erlernten Techniken lassen sich viele weitere Arten der Analyse ableiten, was den Studierenden methodische Flexibilität ermöglicht.

Ergebnisse

Die Durchführung des Seminars im Blended Learning-Format kann als Erfolg gewertet werden. Die Abbruchquote war geringer als in den früheren Semestern und die eingereichten Aufgaben waren meist von hoher Qualität. Im Wintersemester 2021/22 wurde das Seminar

in dieser Form bereits zum zweiten Mal angeboten, wobei die Ergebnisse vergleichbar waren mit dem ersten Semester.

Dieses erfreuliche Resultat scheint zu einem guten Teil auf das Blended Learning-Format an sich zurückzuführen zu sein. Neben räumlicher und zeitlicher Flexibilität beim Erlernen neuer Inhalte, scheint der größte Vorteil darin zu liegen, dass die Videoinhalte beliebig oft angeschaut werden können, nicht verstandene Abschnitte wiederholt werden können und sich Lernende so weniger schnell abgehängt fühlen. Zudem vermittelt das oben beschriebene Echtzeit-Coding (das ebenfalls flexibel konsumierbar ist) in zweierlei Hinsicht ein realistischeres Bild vom Programmieren: Erstens wird deutlich, dass die Bewältigung echter Aufgaben – im Unterschied zu vorgefertigten, kleinteiligen Übungen – einen gewissen zeitlichen Aufwand bedeutet und der Code inkrementell wächst, ohne dass sich jede einzelne Codezeile im Voraus planen ließe. Zweitens wird deutlich, dass es nicht möglich ist, mehr als ein paar Zeilen Code zu schreiben, ohne dass Anpassungen nötig werden bzw. sich Fehler einschleichen. Bei der statischen Darstellung wie in einem Lehrbuch wird jeweils nur fertiger Code, der sich von Beispiel zu Beispiel erweitert, ohne dass der Prozess dahinter sichtbar ist, dargestellt. Die inkrementelle und iterative Realität, die sich beim Echtzeit-Coding zeigt, könnte dazu führen, dass sich Lernende weniger schnell entmutigen lassen, wenn sie bei der eigenen Bearbeitung von Aufgaben auf Probleme stoßen. Gleichzeitig wird in diesen Videos ein rundum lebendiges Bild vom Programmieren gezeigt, was auch die Motivation anregen kann. Insgesamt scheint das Blended Learning-Format geeignet zu sein, um Anfänger:innen aus der Linguistik im Programmieren zu unterrichten.

Als Nachteil muss der Wegfall von physisch präsenten Sitzungen genannt werden, bei denen Teilnehmende gemeinsam Lösungen erarbeiten. Sicher bergen solche Sitzungen die Gefahr, dass einzelne Teilnehmer:innen mit dem Stoff nicht mehr mitkommen, gleichzeitig ist gemeinsames Arbeiten im selben Raum aber auch ein motivierender Faktor [8][9]. Die Blended Learning-Umsetzung des Seminars erfolgte bislang – abgesehen von Übungen im Tutorium – völlig ohne kollaborative Elemente.

Zudem entstand ein gewisses Spannungsverhältnis zwischen den sehr flexibel konsumierbaren Kursinhalten auf der einen Seite und der terminlich festgelegten Abgabe und anschließenden Auswertung der Hausaufgaben – die Auswertung durch den Dozenten erfolgte in einem festen Rhythmus, der abgewartet werden musste. Obschon das Echtzeit-Coding dem Kurs eine gewisse Dynamik verlieh, waren sämtliche Inhalte doch statisch, da die Studierenden nicht mit ihnen interagieren konnten. Dadurch blieb sicher einiges Potential ungenutzt.

5. Ausblick: Programmieren für Geistes- und Sozialwissenschaftler:innen

Aufbauend auf den guten Erfahrungen mit dem Blended Learning-Format zur Vermittlung von Programmierkenntnissen an Linguist:innen wird derzeit ein Lehrangebot konzipiert und entwickelt, das sich neben Sprachwissenschaftler:innen auch an Studierende anderer GSW richtet. Diese Arbeit findet im Rahmen des Projekts *Experimentierraum digitale Medienkompetenz (ExDiMed)* statt. Ziel des Projekts ist es, mithilfe von Blended Learning-Formaten Studierenden aus den GSW digitale Kompetenzen im Umgang mit Medien und Daten an die Hand zu geben. Wie eingangs ausgeführt handelt es sich bei Programmierkenntnissen um eine fundamentale Kompetenz in den zunehmend digitalen GSW. ExDiMed ist wiederum institutionell verankert im virTUos-Verbund und findet so Anschluss an andere Projekte zum Thema **virtuelles** Lehren und Lernen an der **TU Dresden** im **Open Source**-Kontext.

Was als Motivation für die Einführung eines Programmierkurses in der Linguistik galt, gilt in ähnlichem Maße auch für andere GSW, wie anhand von Bedarfsinterviews mit Vertreter:innen verschiedener GSW (konkret: Kommunikations-, Bild-, Geschichts- und Sozialwissenschaft) in Erfahrung gebracht werden konnte: Auch Studierende dieser GSW verfügen in aller Regel über wenige bis keine Programmierkenntnisse. Existierende Datensätze ließen sich zusätzlich zur dominierenden qualitativen Auswertung auch quantitativ analysieren. Darüber hinaus besteht großes Potenzial bei der Erschließung neuer Datenquellen (z.B. über Web Scraping, API-Abrufe oder OCR). Zudem

ist Text nicht nur in der Linguistik, sondern auch in anderen GSW ein wichtiger, wenn nicht der wichtigste Datentypus (z.B. digitalisierte Schriften, Zeitungskorpora oder Social Media-Posts in den Geschichts-, Kommunikations- bzw. Sozialwissenschaften). Der Fokus des Angebots wird also weiterhin auf Textdaten liegen, wobei Exkurse in z.B. flächige Daten (etwa Bilddaten in den Bildwissenschaften) möglich sind.

Die Programmiersprache Python wird beibehalten, neu wird das Angebot aber in der web-basierten IDE *JupyterLab* aufbereitet. In sog. *Jupyter Notebooks* können Code-Zellen mit Markdown-Zellen elegant verwoben werden (Abb. 8 und 9). Letztere können für Erklärtex-te oder die Formulierung von Übungsaufgaben verwendet und als nicht editierbar für Studierende eingestellt werden. Code-Zellen können entweder leer sein (etwa bei Übungsaufgaben) oder bereits Code enthalten, der von den Studierenden erweitert („Schreibe den Code fertig“), modifiziert („Setze relevante Parameter ein“ bzw. „Finde den Fehler“) und in jedem Fall ausgeführt werden kann. Insgesamt lässt sich so ein interaktives Notebook gestalten, in dem sich Studierende selbstgesteuert Wissen aneignen und dieses im gleichen Zug anwenden können. Studierende erhalten ihre persönliche Kopie eines Notebooks und können deshalb nicht nur wo explizit vorgesehen Code zu einer Übung einfügen, sondern auch beliebig viele weitere Codezellen sowie Markdown-Zellen hinzufügen (um eigene Ideen zu verfolgen bzw. um an relevanten Stellen eigene Notizen festzuhalten).

Die Unterteilung von Code in Zellen (ggf. mit Markdown-Zellen dazwischen) vermittelt und unterstreicht den bereits als wichtig vorgestellten inkrementellen Charakter von Programmieren: Zelle für Zelle (Schritt für Schritt) wird ein Problem gelöst und bei jeder Zelle wird überprüft, ob der Code das gewünschte (Zwischen-)Resultat liefert. Dadurch wird auch die beim algorithmischen Denken so zentrale schrittweise Lösungssuche praktiziert (s. Kap. 1).

Die Notebooks werden die Videos beim bisherigen Format größtenteils ablösen. Einzelne Notebooks, etwa zur Einführung in die IDE, sollen aber weiterhin von Videos flankiert wer-

den. Das bewährte Echtzeit-Coding soll beibehalten und mit Blick auf eine konsistente IDE in JupyterLab umgesetzt werden. Auch diese Videos lassen sich elegant in die Notebooks einbetten.

Die oben erwähnten Vorteile des Blended Learning-Formats (individuelles Lerntempo und Ansprechen verschiedener Kompetenzstufen, zeitlich und örtlich flexibles Lernen, Vermittlung eines realistischen Bildes von Programmieren) gelten auch bei einer Verschiebung weg von Video mit Skript hinzu interaktiven, teils von Videos flankierten Notebooks. Gleichzeitig sind die Inhalte wesentlich dynamischer aufbereitet und Studierende können unmittelbar mit ihnen interagieren, was das Angebot im Vergleich zum bisherigen weniger statisch macht.

Um zusätzlich Interaktion *zwischen* den Studierenden zu erreichen sind thematisch eingegrenzte *Hackathons* geplant, in deren Rahmen Teilnehmer:innen innerhalb einiger Stunden kollaborativ ein Problem bearbeiten. Ebenso sind Übungsaufgaben in Gruppenarbeit angedacht, wobei hierfür die Kollaborationsmöglichkeit bei JupyterLab genutzt werden soll (Voraussetzung ist, dass die Studierenden auf denselben Server zugreifen, wodurch auch kollaboratives, synchrones Arbeiten von getrennten Orten aus möglich ist). Kollaborative Lern-Settings haben sich in verschiedenen Studien als förderlich bei der Vermittlung von Programmierkenntnissen erwiesen [11-13]. In solch einer Umgebung können Studierende Ideen und Lösungen ko-konstruieren, die sie isoliert womöglich nicht entwickelt hätten [12]. Inhaltlich wird zwischen einem Grundlagen- und mehreren Aufbaumodulen unterschieden. In den Notebooks des Grundlagenmoduls soll wie bislang das theoretische Fundament zum Programmieren mit Python gelegt werden. Themen wie Variablen, Operatoren, Datentypen, Kontrollstrukturen (Bedingte Anweisungen und Schleifen), Funktionen und Methoden, Import von Modulen und Packages sowie Input/Output werden erklärt, wobei wiederum auf die realitätsnahe praktische Anwendung in Codebeispielen und Übungsaufgaben Wert gelegt wird. Noch vor die Grundlagenvermittlung soll eine knappe, angewandte Einführung in algorithmisches Denken (s. Kap. 2) gestellt wer-

den. Die Verinnerlichung dieser grundsätzlichen Art der Herangehensweise an ein (Programmier-)Problem ist nämlich wichtiger als die konkrete Syntax und Semantik der jeweiligen Programmiersprache [4, S. 90]. Ergebnisse aus einem iterativ konzipierten Programmier-

kurs [11], in dem jeweils nur ein Parameter zur vorangegangenen *condition* verändert wurde, legen nahe, dass das Voranstellen selbst einer kurzen Einheit zu algorithmischem Denken zu besseren Programmierfertigkeiten führt.

Bedingte Anweisungen

Wir widmen uns als Erstes den bedingten Anweisungen und definieren dafür einen simplen string. Führe wie immer die Zelle aus, um `sentence` zu initialisieren.

```
sentence = "Der morgige Tag wird schön."
```

Eine bedingte Anweisung wird mit `if` eingeleitet, z.B.:

```
if sentence.startswith("Der"):
    print("Der Satz fängt mit einem Artikel im Maskulinum an.")
```

Natürlichsprachlich formuliert liest sich der obige Code: "Wenn der Satz mit 'Der' anfängt, dann geben wir '...' zurück".

Übrigens haben wir gerade eine sog. string-Methode kennengelernt, nämlich `startswith`, die überprüft, ob ein string (hier: `sentence`) mit der in der Klammer definierten Zeichenkette beginnt. Diese und andere Methoden besprechen wir im Detail im nächsten Notebook.

Wie gesagt, bei bedingten Anweisungen geht es immer um `True` oder `False`. Die obige Bedingung ist eigentlich abgekürzt formuliert. Ausformuliert lautet sie:

```
if sentence.startswith("Der") == True:
    print("Der Satz fängt mit einem Artikel im Maskulinum an.")
```

Nur wenn die angegebene Bedingung zutrifft (also den Wert `True` annimmt), wird der im Anweisungskörper geschriebene Code ausgeführt. In der Praxis verwendet man stets die Abkürzung, die, wie oben gezeigt, auch natürlichsprachlich intuitiv Sinn ergibt.

Nun ändern wir den Satz (genauer gesagt: wir referenzieren mit der Variablen `sentence` ein neues Objekt, das alte Objekt verliert seine Referenz):

```
sentence = "Die morgige Nacht wird schön."
```

Verwenden wir nun die gleiche bedingte Anweisung, so geschieht nichts, denn sie ergibt `False` und folglich wird die Anweisung im Körper nicht ausgeführt. Wir brauchen also eine zweite Bedingung, zusätzlich zur ersten, die wir beibehalten möchten:


Abb. 8: Einführung in bedingte Anweisungen als Beispiel für die Kombination aus Markdown-Zellen (Erklärtexte) und (hier nicht ausgeführten) Code-Zellen in JupyterLab.

Anschließend an das Grundlagenmodul sollen sich Aufbaumodule unterschiedlichen Anwendungsfällen widmen. Darunter fallen bisherige anwendungsorientierte Inhalte (etwa Web

Scraping), die so modifiziert werden, dass sie auch für Studierende anderer GSW von Interesse sind. Zusätzlich sind fachspezifische Module geplant, die konkret auf die Bedarfe der

jeweiligen GSW eingehen (als Beispiel: automatisierte *News Factor*-Analyse für die Kommunikationswissenschaft). Ein weiteres Aufbau-Modul soll statistische Auswertung sowie Visualisierungsmöglichkeiten behandeln. Alle Aufbaumodule sollen voneinander unabhängig

sein, sodass Studierende aus den verschiedenen GSW unter der Voraussetzung des theoretischen Fundaments modular einen für sie relevanten Programmierkurs zusammenstellen können.

 **Übung:** Bei Variablennamen haben wir ja schon gelernt, dass diese case-sensitive sind. Finde nun heraus, ob dies auch bei Zeichenketten der Fall ist, indem Du zwei Variablen mit zwei Zeichenketten definierst, die sich nur in punkto Groß-/Kleinschreibung unterscheiden. Lass Dir anschließend ausgeben, ob die Werte der beiden Variablen für Python gleich sind oder nicht.

```
# In diese Zelle kannst Du den Code zur Übung schreiben.  
str1 = "hallo"  
str2 = "HALLO"  
print(str1 == str2)
```

False

Abb. 9: Kombination aus Markdown-Zelle (Übungsbeschreibung) und (ausgeführter) Code-Zelle in JupyterLab.

Ungeachtet des angepassten und erweiterten Formats bleibt der Anspruch bestehen, dass Studierende mit wenigen bis keinen Vorkenntnissen durch dieses Lehrangebot in die Lage versetzt werden, geistes- und sozialwissenschaftliche Fragestellungen auf digital kompetente Weise zu bearbeiten – vom Beschaffen eigener Daten über die Datenaufbereitung bis hin zur Auswertung und Visualisierung. Mittelfristig soll das entwickelte Lehrangebot im neuen Masterstudiengang Digital Humanities an der TU Dresden (ab Wintersemester 2022/23) verankert werden. Zudem soll es als Open Educational Ressource bereitgestellt und in entsprechenden Verzeichnissen indiziert werden.

Danksagung

Wir bedanken uns bei Gregor Mitzka, der sowohl den bisherigen Programmierkurs als auch ExDiMed als studentische Hilfskraft unterstützt.

Das Projekt ExDiMed wird von der Stiftung Innovation in der Hochschullehre im Rahmen des virTUos-Verbands gefördert.

Literatur

- [1] Baum, C. & Stäcker, Th, Die Digital Humanities im deutschsprachigen Raum. Methoden. Theorien. Projekte, in: Dies. (Hg.): Grenzen und Möglichkeiten der Digital Humanities (= Sonderband der Zeitschrift für digitale Geisteswissenschaften 1), 2016. https://doi.org/10.17175/sb001_023
- [2] Snee, H., Hine, C., Morey, Y., Roberts, S., & Watson, H. Digital Methods as Mainstream Methodology. An Introduction. In: H. Snee, C. Hine, Y. Morey, S. Roberts, & H. Watson (Hrsg.), Digital Methods for Social Science. An Interdisciplinary Guide to Research Innovation; S. 1–11. Palgrave Macmillan UK, 2016. https://doi.org/10.1057/9781137453662_1
- [3] Deutsche Forschungsgemeinschaft. Digitaler Wandel in den Wissenschaften. Impulspapier, 2020. <https://doi.org/10.5281/zenodo.4191345>
- [4] Jannidis, F.: Grundbegriffe des Programmierens In: Ders., Kohle, H., & Rehbein, M. (Hrsg.). Digital Humanities. Eine Einführung. Metzler, 2017, 68-95.
- [5] Seufert, S., Mayr, P. Blended Learning. In: Fachlexikon e-learning. Wegweiser durch das e-Vokabular. Bonn, 2002.
- [6] Schoop, E., Bukvova, H., Lieske, C. Blended-Learning arrangements for higher education in the changing knowledge society. In: Proceedings of the International Conference on Current Issues in Management of Business and Society Development, 2010. <https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa-26183>
- [7] Kerres, M. Mediendidaktik Konzeption und Entwicklung mediengestützter Lernangebote. München: Oldenbourg Wissenschaftsverlag, 2013. <https://doi.org/10.1524/9783486736038>

- [8] Loes, Chad N. The Effect of Collaborative Learning on Academic Motivation. In: Teaching & Learning Inquiry 10(1), 2022. <https://doi.org/10.20343/teachlearningqu.10.4>
- [9] Haaranen, L. Programming as a Performance. Live-streaming and Its Implications for Computer Science Education. In: ITiCSE '17: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education. June 2017, S. 353-358. <https://doi.org/10.1145/3059009.3059035>
- [10] Pirker, J., Steinmaurer, A & Karakas, A. Beyond Gaming. The Potential of Twitch for Online Learning and Teaching, In: 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2021), 2021, S. 74-80. <https://doi.org/10.1145/3430665.3456324>
- [11] Koulouri, T., Lauria, S., & Macredie, R. D. Teaching Introductory Programming. A Quantitative Evaluation of Different Approaches, ACM Transactions on Computing Education, 2015, 14 (4), S. 1-28. <https://doi.org/10.1145/2662412>
- [12] Beck, L., & Chizhik, A. Cooperative learning instructional methods for CS1. Design, implementation, and evaluation, ACM Transactions on Computing Education, 13(3), 2013, 10:1-10:21. <https://doi.org/10.1145/2492686>
- [13] Braught, G., Wahls, T., & Eby, L. M. The Case for Pair Programming in the Computer Science Classroom. ACM Transactions on Computing Education, 11(1), 2011, 2:1-2:21. <https://doi.org/10.1145/1921607.1921609>