Lessons Learned 2, 1 (2022)
Submitted: 30.03.2022
Accepted: 19.05.2022
DOI: https://doi.org/10.25369/ll.v2i1.37
ISSN: 2749-1293 (print); 2749-1307 (online).

Digital skills for humanities and social science students. Benefits of a Blended Learning format for teaching programming skills

Y. Frommherz*, J. Langenhorst

Chair of Applied Linguistics, Institute of German Studies, Dresden University of Technology

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 niedrigschwellige 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 GSWs 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 digitizing 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 are of limited use in that they predefine how researchers can interact with their data. 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 proven 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. Introduction

The humanities and social sciences (hereafter: *HSS*) are undergoing a *digital turn* [1-3] It has never been easier to access large amounts of data relevant to the HSS – whether from the Internet or via the digitization of analog data sets. This development makes the use of quantitative methods possible and at the same time necessary [4, p. 297]. However, the focus in HSS has traditionally been on qualitative research. In addition to data access techniques, these newly relevant quantitative methods go beyond the skills typically taught in HSS curricula. At the same time, however, low-threshold tools have emerged that can be used to accomplish many new tasks within ever more digital HSS research. For example, there is easy-to-use software for examining texts with regard to patterns occurring in them (such as keywords). However, the analytical categories that a user can choose between are always predefined and thus limited. As soon as research questions are pursued for which these tools were not designed, at least basic programming skills are required. Thus, if students and researchers rely solely on such software, they will hardly be able to exploit the full potential which arises from the digital turn.

The Chair of Applied Linguistics at TU Dresden therefore believes that teaching digital skills in general and programming skills in particular is a necessary part of a linguistics program. In order to sustainably prepare students for the increasingly digital research and professional world, the Chair has been offering a programming course for many years.

One challenge in teaching linguistics students programming is that the students generally have no prior knowledge of programming and little technical knowledge in general. Therefore, from the beginning, the design of the programming course has focused on how these skills can best be taught to the given target audience. With this question in mind, the course has been and continues to be developed. Among other things, it was converted from a traditional face-to-face seminar to a Blended Learning format. In such a setting, elements of traditional, temporally synchronous and spatially co-present teaching are combined with online learning content of different media types that students can consume temporally asynchronously and spatially flexibly [5-7]. This allows students to largely control their own learning progress, which has advantages when learning programming (see below).

This paper provides a detailed report on how programming skills can be taught to students of German Philology, and specifically Applied Linguistics. Building on a detailed argument of how HSS researchers benefit from programming skills as a fundamental digital competence (Ch. 2), the initial situation leading to the switch to a Blended Learning format is described (Ch. 3). Chapter 4 delineates the current programming course and, finally, Chapter 5 concludes with an outlook on an ongoing research project that will further optimize the existing course and extend it to students of other HSS. The article is not least addressed to interested lecturers within the HSS who also want to prepare their students for the possibilities and requirements arising from the digital turn.

2. Why do humanities and social science students need programming skills?

In the following, concrete steps of an exemplary study will be used to demonstrate why HSS researchers benefit from programming skills. Where appropriate, reference is made to related techniques that also require basic programming skills, but in return provide researchers with methodological flexibility. The exemplary study will use a quantitative approach to investigate how so-called success coaches on YouTube engage their audience linguistically. This is a topic that could be investigated as part of a student paper in a linguistics program. Typically, this would be done in a qualitative way, i.e., the focus would be on individual pieces of evidence (individual instances of *audience engagement*) that would be subjected to in-depth analysis.

In the quantitative approach pursued here, however, a large amount of data is to be analyzed automatically in order to be able to recognize patterns that go beyond individual pieces of evidence. Ideally, this will reveal regularities that would remain undiscovered using a qualitative approach. In any case, however, the patterns in the data can be *quantified,* allowing clear statements about the relevance of a certain phenomenon in the given context. In the first step, the data, i.e., the language occurring in the videos, needs to be accessed. With the help of existing modules (roughly speaking, ready-made code blocks for a specific purpose) for the programming language Python used here, as well as a few additional lines of code, subtitles of a given YouTube video can be accessed using its ID (each YouTube video has such an ID). The subtitles can then be saved in a suitable format including relevant metadata (Fig. 1 and Fig. 2).

```
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
```

Fig. 1: Part of the code for downloading YouTube transcripts.

```
{
    "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",
```

Fig. 2: Saved YouTube transcripts.

Similar modules that allow researchers to download specific content from social media with just a few lines of code are available for various platforms. Apart from this, researchers can also *scrape* themselves, i.e., download specific content from almost any website (so called *web scraping*). Another data mining method available to researchers with programming skills is *Optical Character Recognition* (*OCR*). This makes it possible to read characters from images, making the extracted text machine-readable, which is interesting both for historical sources, but also for so-called *sharepics* (images with text for sharing, such as on Instagram).
The second step is to preprocess the collected data. Using suitable modules, the data can be managed, filtered and prepared for analyses in a flexible and efficient way. For the analysis in the context of the exemplary study, the collected data should be available word for word, which is done in a step called tokenization (Fig. 3).

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

Fig. 3: Tokenized text.

In the third step, the data is analyzed. For this purpose, *audience address* is operationalized as collocations of the personal pronouns *Du* (informal 'you' in German) and *Sie* (formal 'you'). Collocations are words that frequently occur together, such as *heavy* and *rain* in common speech. Here, we will analyze which words occur more frequently in the vicinity of the addressing pronouns *Du/Sie* than can be expected from their frequency in the rest of the text (Fig. 4. shows this for *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

Fig. 4: Collocations of ‚Du'.

For *Du,* Fig. 4 shows, among other things, that the modal verb *können* ('can') occurs most frequently (compared to the expected frequency) in the analyzed YouTube videos of success coaches. It seems that the proverbial key to success advertised by the examined coaches lies in the intrinsic ability of the target audience. This hypothesis could now be a starting point for further quantitative and qualitative investigation.

In summary, as demonstrated by this use case, only with programming skills researchers can efficiently access relevant data, preprocess and analyze it in a methodologically flexible manner tailored to the research question at hand. Furthermore, programming skills are immensely useful for visualizing data. Although no-code software for this purpose exists as well (e.g., SPSS or Microsoft Office software), programming skills enable a much more flexible visualization, which is especially advantageous for the purpose of scientific publication.

In addition to these concrete methodological benefits of programming skills, digital work also enables researchers to acquire or consolidate *algorithmic thinking* [4, p. 95]. Algorithmic thinking is the ability to 1) analyze a given problem with precision, 2) define simple steps to solve the problem, 3) create a solution from these steps, i.e., construct a complete and correct algorithm, 4) test this algorithm for all typical as well as atypical cases, and 5) improve its efficiency. Algorithmic thinking is not only a prerequisite for successful programming, but it equally grows out of exposure to code. Researchers also benefit from practicing algorithmic thinking outside of programming, for example in their scientific work, as a thorough, sophisticated approach to a problem (1-3 above) as well as the constant effort to test a potential solution (4) are elementary aspects in both cases.

3. Initial situation

Being able to program is thus an increasingly relevant digital skill for students. Therefore, the Chair of Applied Linguistics has long offered a course called *Programming for Linguists*. As mentioned, before the switch to Blended Learning, the course was conducted as a traditional face-to-face seminar teaching the scripting language Perl. From week to week, new content was taught in the seminar and weekly assignments had to be solved by the students. In addition, a tutorial was offered, which is unusual for seminars in linguistics, but proved to be necessary due to the density of the content. Individual seminar sessions were used to collaboratively work on assignments, while others served as an introduction to programming concepts or, later in the semester, to specific linguistic techniques such as *web scraping* and, in particular, corpus analysis methods.

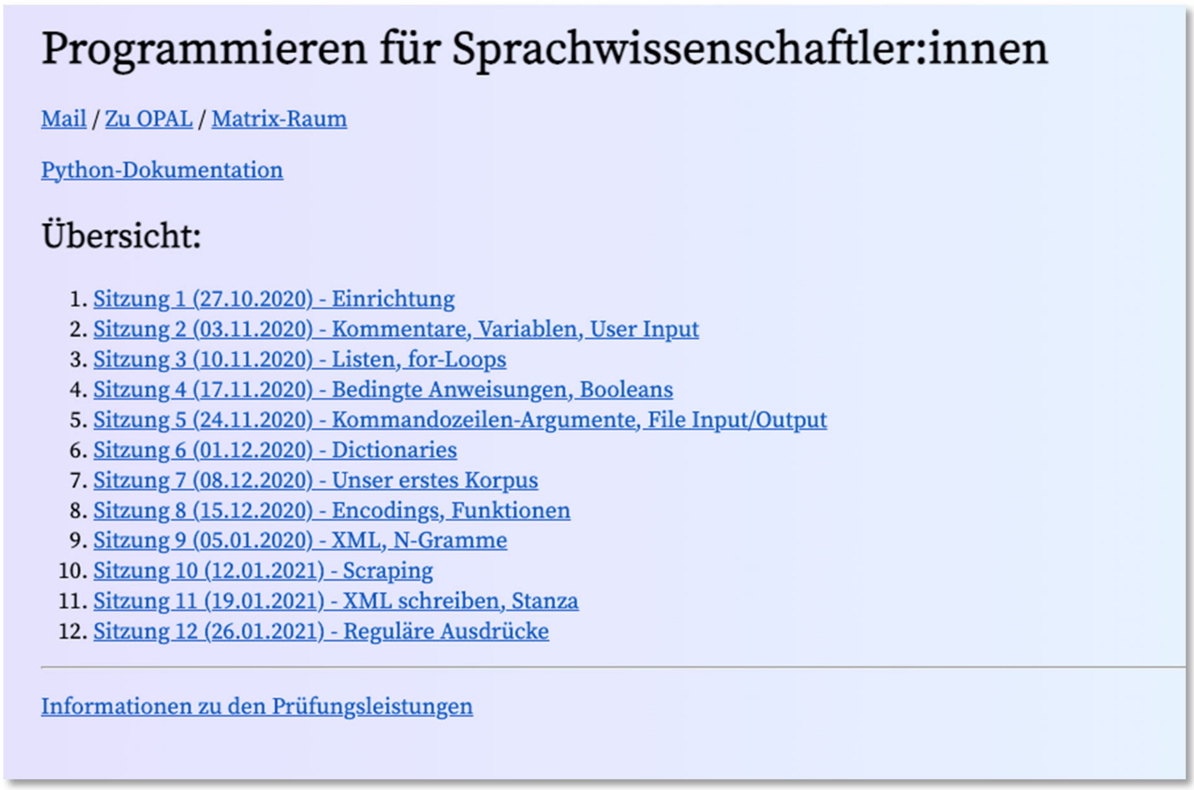
Over the course of the semesters, many students were inspired to go about solving their research questions using programming skills. However, at the same time, many participants did not complete the seminar – the dropout rate was significantly higher compared to other seminars at the Chair. Teaching a variety of programming techniques in a single semester to students most of whom had never programmed before and often had little prior technical knowledge turned out to be challenging in this mode.

4. Switch to Blended Learning

Starting with the winter semester 2020/21, the course was first conducted in a Blended Learning format as part of the Covid-19-related conversion to digital teaching. On the one hand, asynchronous video content was offered. On the other hand, the tutorial continued to be held synchronously and – as far as the pandemic allowed – also spatially co-present. Teaching materials of various media types, first and foremost the aforementioned videos, but also code snippets, illustrative diagrams, supplementary documents, and links to further resources were bundled (Fig. 5 and Fig. 6). Communication in the seminar took place via a chat room. Further advice on problems was primarily given via e-mail. Later in the semester, so-called *real-time coding* was made available in video format, i.e., recordings in which complete programming tasks were solved in real time, such as scraping a web page. Here, the individual steps – analyzing the page, finding links, downloading pages, extracting and annotating text and metadata, assembling a corpus file – were demonstrated step by step, rather than merely explained (Fig. 7). These real-time coding sessions were inspired by coding streams on the streaming platform

Twitch and on YouTube, where developers can be watched programming live – or afterwards in case the stream is recorded. Such streams enjoy great popularity and now represent a kind of teaching format in its own right [9][10].

One difference between the real-time coding used in the seminar and the Twitch streams is, of course, that there is never any live interaction with the viewers.



Programmieren für Sprachwissenschaftler:innen

Mail / Zu OPAL / Matrix-Raum

Python-Dokumentation

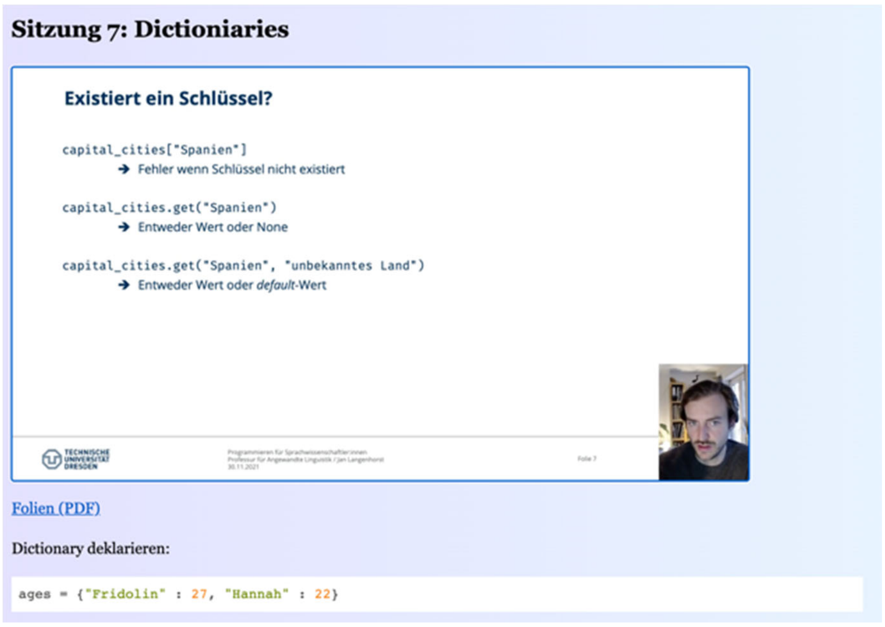
Übersicht:

1. Sitzung 1 (27.10.2020) - Einrichtung
2. Sitzung 2 (03.11.2020) - Kommentare, Variablen, User Input
3. Sitzung 3 (10.11.2020) - Listen, for-Loops
4. Sitzung 4 (17.11.2020) - Bedingte Anweisungen, Booleans
5. Sitzung 5 (24.11.2020) - Kommandozeilen-Argumente, File Input/Output
6. Sitzung 6 (01.12.2020) - Dictionaries
7. Sitzung 7 (08.12.2020) - Unser erstes Korpus
8. Sitzung 8 (15.12.2020) - Encodings, Funktionen
9. Sitzung 9 (05.01.2020) - XML, N-Gramme
10. Sitzung 10 (12.01.2021) - Scraping
11. Sitzung 11 (19.01.2021) - XML schreiben, Stanza
12. Sitzung 12 (26.01.2021) - Reguläre Ausdrücke

Informationen zu den Prüfungsleistungen

Fig. 5: Overview page of the course (winter semester 2020/21).

Apart from the new format, another innovation was that Python was taught instead of Perl, since by the time the seminar was revised, Python had advanced to be *the* programming language in science. Python offers numerous libraries for analyzing structured data (including Pandas), data visualization (including Matplotlib), annotation (including the NLP libraries Stanza, SpaCy, NLTK, TextBlob, etc.), and scraping (including BeautifulSoup, Selenium). Additionally, Python is potentially easier to learn for beginners than previously-taught Perl due to its somewhat simpler syntax and semantics (for a comparison of Python/Java: [11]).
Even though existing libraries are one reason to teach a specific language, the focus of the seminar – as a *programming seminar* – was to have students develop as much code as possible themselves. Students should not simply assemble existing code using Python modules like in a construction kit, but rather work out as many steps as possible themselves. Pure introductions to libraries would hardly have added any value compared to the existing low-threshold tools mentioned at the beginning, which also exist for the analysis of (corpus) linguistic data (e.g., Corpus Workbench, Sketch Engine). It would be possible, for example, to have a Python package output frequent co-occurences with just a few lines of code. With little effort, however, one can reproduce this process, making transparent how the data is handled. Thus, one is not confronted with a black box in the form of a tool, but, having devised the individual steps, one understands the process better.
After the setup of Python was mastered and the students had familiarized themselves with the shell, the course introduced the basics of programming such as control structures and data types as well as related simple exercises. Midway through the semester, more and more specific corpus linguistic applications were worked on.



Sitzung 7: Dictioniaries

Existiert ein Schlüssel?

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

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

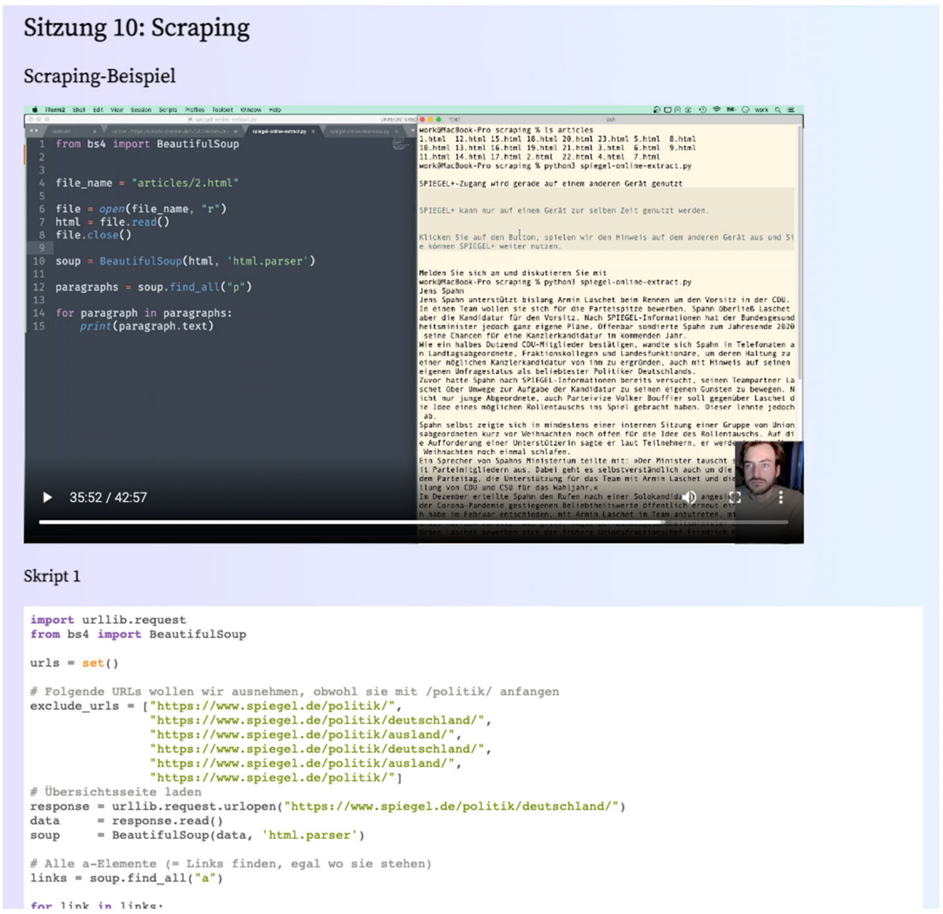
```
capital_cities.get("Spanien", "unbekanntes Land")
```
➔ Entweder Wert oder *default*-Wert

Folien (PDF)

Dictionary deklarieren:

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

Fig. 6: Course content.



Sitzung 10: Scraping

Scraping-Beispiel

```
from bs4 import BeautifulSoup


file_name = "articles/2.html"

file = open(file_name, "r")
html = file.read()
file.close()

soup = BeautifulSoup(html, 'html.parser')

paragraphs = soup.find_all("p")

for paragraph in paragraphs:
    print(paragraph.text)
```

```
work@MacBook-Pro scraping % ls articles
1.html  12.html 15.html 18.html 20.html 23.html 5.html  8.html
10.html 13.html 16.html 19.html 21.html 3.html  6.html  9.html
11.html 14.html 17.html 2.html  22.html 4.html  7.html
work@MacBook-Pro scraping % python3 spiegel-online-extract.py

SPIEGEL+-Zugang wird gerade auf einem anderen Gerät genutzt


SPIEGEL+ kann nur auf einem Gerät zur selben Zeit genutzt werden.


Klicken Sie auf den Button, spielen wir den Hinweis auf dem anderen Gerät aus und Si
e können SPIEGEL+ weiter nutzen.



Melden Sie sich an und diskutieren Sie mit
work@MacBook-Pro scraping % python3 spiegel-online-extract.py
Jens Spahn
Jens Spahn unterstützt bislang Armin Laschet beim Rennen um den Vorsitz in der CDU.
In einen Team wollen sie sich für die Parteispitze bewerben. Spahn überließ Laschet
aber die Kandidatur für den Vorsitz. Nach SPIEGEL-Informationen hat der Bundesgesund
heitsminister jedoch ganz eigene Pläne. Offenbar sondierte Spahn zum Jahresende 2020
 seine Chancen für eine Kanzlerkandidatur im kommenden Jahr.
Wie ein halbes Dutzend CDU-Mitglieder bestätigen, wandte sich Spahn in Telefonaten a
n Landtagsabgeordnete, Fraktionskollegen und Landesfunktionäre, um deren Haltung zu
einer möglichen Kanzlerkandidatur von ihm zu ergründen, auch mit Hinweis auf seinen
eigenen Umfragestatus als beliebtester Politiker Deutschlands.
Zuvor hatte Spahn nach SPIEGEL-Informationen bereits versucht, seinen Teampartner La
schet über Umwege zur Aufgabe der Kandidatur zu seinen eigenen Gunsten zu bewegen. N
icht nur junge Abgeordnete, auch Parteivize Volker Bouffier soll gegenüber Laschet d
ie Idee eines möglichen Rollentauschs ins Spiel gebracht haben. Dieser lehnte jedoch
 ab.
Spahn selbst zeigte sich in mindestens einer internen Sitzung einer Gruppe von Union
sabgeordneten kurz vor Weihnachten noch offen für die Idee des Rollentauschs. Auf di
e Aufforderung einer Unterstützerin sagte er laut Teilnehmern, er werde
 Weihnachten noch einmal schlafen.
Ein Sprecher von Spahns Ministerium teilte mit: »Der Minister tauscht s
it Parteimitgliedern aus. Dabei geht es selbstverständlich auch um die
dem Parteitag, die Unterstützung für das Team mit Armin Laschet und die
llung von CDU und CSU für das Wahljahr.«
Im Dezember erteilte Spahn den Rufen nach einer Solokandida   angesi
der Corona-Pandemie gestiegenen Beliebtheitswerte öffentlich erneut ei
h habe im Februar entschieden, mit Armin Laschet im Team anzutreten, mi

Neben Laschet bewerben sich der frühere Unionsfraktionschef Friedrich M
```

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/"]
# 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:
```

Fig. 7: Real-time coding video with associated code snippet.

Corpus linguistics studies linguistic phenomena on the basis of large collections of texts (*corpora*), which are annotated linguistically (e.g., with parts of speech) and with metadata (e.g., author, year, genre). Students were first provided with differently structured corpora to be analyzed for frequent words, n-grams (*n* consecutive words), etc. Then, the focus shifted to creating and annotating own corpora from web sources. Overall, students learned to write scripts to examine self-created corpora for various linguistic patterns as well as the factors that condition their occurrence. Many other types of analyses can be derived from the techniques learned, providing students with methodological flexibility.

5. Results

The implementation of the seminar in the Blended Learning format can be considered a success. The dropout rate was lower than in previous semesters and the submitted assignments were mostly of high quality. In the winter semester 2021/22, the seminar was offered in the same form for the second time, with results comparable to the first semester.

This pleasing result seems to be due in large part to the Blended Learning format itself. In addition to spatial and temporal flexibility when learning new content, the greatest advantage seems to be that the video content can be watched as often as desired and sections that are not understood well can be repeated resulting in learners being less likely to feel disengaged. In addition, the real-time coding described above (which can also be consumed flexibly) conveys a more realistic image of programming in two respects: First, it becomes clear that accomplishing real tasks – as opposed to prefabricated, small-scale exercises – requires a certain amount of time, and that code grows incrementally in a way where not every single line of code can be planned in advance. Second, it becomes clear that it is not possible to write more than a few lines of code without adjustments becoming necessary or errors creeping in. In contrast, in static representations like textbooks only finished code is shown. The code may be extended from example to example, but the process behind it never becomes visible in a dynamic way. The incremental and iterative reality experienceable in real-time coding could result in learners being less likely to feel discouraged when they encounter problems in their own work. At the same time, the videos present an all-round vivid picture of programming, which can also stimulate learners' motivation. Overall, the Blended Learning format seems to be suitable for teaching programming to beginners from linguistics.

A disadvantage is the omission of physically copresent sessions in which participants work out solutions together. Certainly, such sessions bear the risk that individual participants no longer keep up with the material, but at the same time, working together in the same room is also a motivating factor for students [8][9]. The Blended Learning implementation of the seminar has so far been conducted completely without collaborative elements – apart from exercises in the tutorial. In addition, a certain tension arose between the very flexibly consumable course content on the one hand and the strictly scheduled submission and subsequent evaluation of the homework – the evaluation by the lecturer took place in a fixed rhythm. Although the real-time coding videos gave the course a certain dynamic, all content was static in that the students could not interact with it. This certainly left some potential unused.

6. Outlook: Programming for humanities and social sciences

Building on the positive experience with the Blended Learning format for teaching programming skills to linguists, a course offering is currently being designed and developed that aims not only at linguists but also students from other HSS. This work is conducted as part of the *Experimentierraum Digitale Medienkompetenz* (*ExDiMed*) project. The aim of the project is to provide students from HSS with digital skills in dealing with media and data by means of Blended Learning formats. As stated at the beginning, programming is a fundamental skill in the increasingly digital HSS. ExDiMed, in turn, is institutionally anchored in the virTUos network and thus linked with other projects on the topic of **vir**tual teaching and learning at **TU** Dresden in an **o**pen-**s**ource context.

The motivation for introducing a programming course in linguistics also applies to other HSS, as could be ascertained conducting interviews with representatives of various HSS (specifically: communication, visual, historical, and social sciences): Students in these HSS generally also have little to no prior programming knowledge. In addition to the dominant qualitative evaluation, existing data sets from these fields could also be analyzed quantitatively.

Furthermore, there is great potential in tapping new data sources (e.g., via web scraping, API retrievals, or OCR). Moreover, text is an important, if not the most important, data type not only in linguistics, but also in other HSS (e.g., digitized historical writings, newspaper corpora, or social media posts in history, communication, and social sciences, respectively). The focus of the offering will thus remain on textual data, with possible excursions into, for example, two-dimensional data (such as images in the visual sciences).

The programming language Python is retained, but it will now be taught in the web-based IDE *JupyterLab.* In so-called *Jupyter Notebooks,* code cells can be elegantly interwoven with markdown cells (Fig. 8 and 9). The latter can be used for explanatory texts or the formulation of exercises and can be set as non-editable for students. Code cells can either be empty (e.g. for exercise tasks) or already contain code that can be expanded by students ("Finish the code"), modified ("Set relevant parameters" or "Find the error") and, in any case, executed.

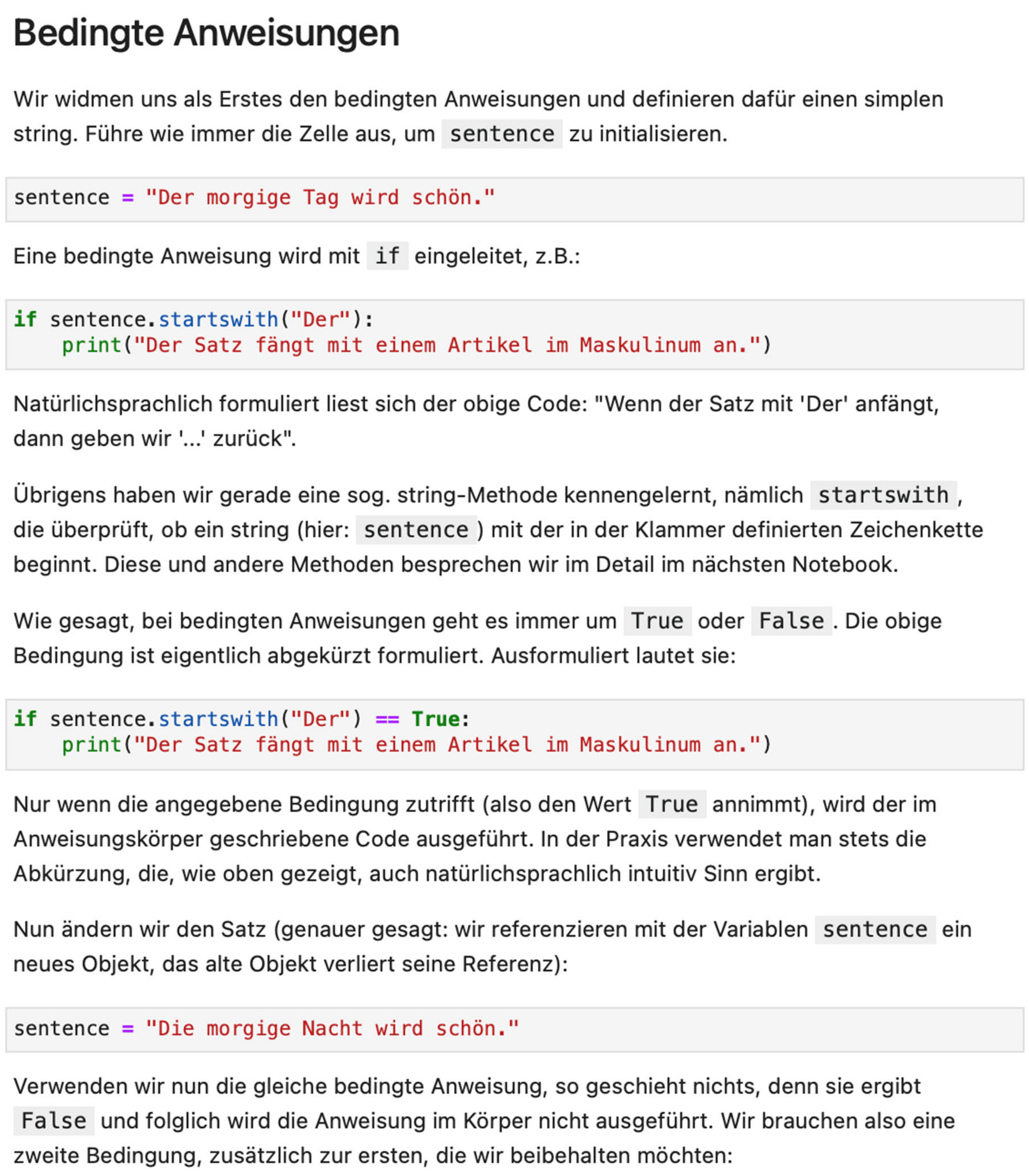
Hence, an interactive notebook can be designed in which students can acquire knowledge in a self-directed manner and apply it at the same time. Students receive their personal copy of a notebook and can therefore not only add code to an exercise where explicitly expected, but also add as many additional code and markdown cells as they wish (e.g., to pursue their own ideas or to make notes at relevant points).

The division of code into cells (possibly with markdown cells in between) conveys and underlines the incremental character of programming already emphasized as important: Cell by cell (step by step) a problem is solved, and in each cell it is checked whether the code delivers the desired (intermediate) result. In this way, the step-by-step approach which is so central to algorithmic thinking is also practiced (see Ch. 1). The notebooks will largely replace the videos from the current format. However, individual notebooks, for example for the introduction to the IDE, will be accompanied by videos. The well-received real-time coding videos will also be retained. Videos can be embedded elegantly in the notebooks.

The advantages of the Blended Learning format mentioned above (individual learning pace and addressing different levels of competence, flexible learning in terms of time and location, conveying a realistic picture of programming) also apply after shifting from a video-based approach to interactive notebooks which are partly accompanied by videos. At the same time, the content is prepared much more dynamically, and students can interact with it directly, which makes the course less static compared to the current one.

In order to achieve additional interaction *between the* students, *hackathons* are planned, in which participants work collaboratively on a certain problem within a few hours. Exercises in group work are also planned, for which the collaboration option inside JupyterLab may be used (the prerequisite is that the students access the same server, which also enables collaborative, synchronous work from separate locations). Collaborative learning settings have been shown by various studies to be beneficial in teaching programming skills [11-13]. In such an environment, students can co-construct ideas and solutions that they may not develop in isolation [12].

In terms of content, a distinction is made between a basic module and several advanced modules. In the notebooks of the basic module, the theoretical foundation for programming with Python will be laid. Topics such as variables, operators, data types, control structures (conditional statements and loops), functions and methods, import of modules and packages as well as input/output are explained, again putting emphasis on hands-on application and realistic exercises. The basic module will be preceded by a concise, applied introduction to algorithmic thinking (see Ch. 2). Internalizing this basic way of approaching a (programming) problem may be even more important than the concrete syntax and semantics of a given programming language [4, p. 90]. Results from an iteratively designed programming course [11] in which only one parameter was changed from condition to condition suggest that offering even a short unit on algorithmic thinking prior to teaching actual coding leads to better programming skills.



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.

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

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

```python
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:

```python
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.

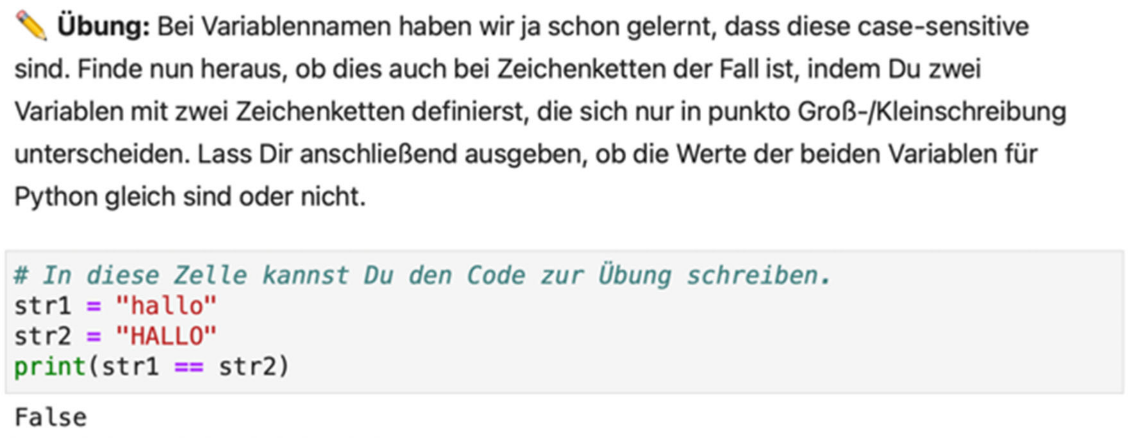
Nun ändern wir den Satz (genauer gesagt: wir referenzieren mit der Variablen `sentence` ein neues Objekt, das alte Objekt verliert seine Referenz):

```python
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:

Fig. 8: Introduction to conditional statements as an example of the combination of markdown cells (explanatory texts) and code cells (not executed here) in JupyterLab.

Following the basic module, advanced modules will be dedicated to different use cases. These include applied contents from the current course (such as web scraping), which will be modified in such a way that they are also of interest to students from other HSS. In addition, subject-specific modules are planned that specifically address the requirements of the respective HSS (as an example: automated *news factor analysis* for communication science). Further advanced modules will deal with statistical analysis and visualization. All advanced modules will be independent of each other, so that students from different HSS can compile a programming course relevant to them on a modular basis, presupposing they have completed the basic module. Regardless of the adapted and expanded format, the goal remains that this course should enable students with little to no prior knowledge to work on HSS research questions in a digitally competent manner – from accessing their own data to data preprocessing, analysis, and visualization.


✏️ **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
```

Fig. 9: Combination of markdown cell (exercise description) and (executed) code cell in JupyterLab.

In the medium term, the developed course is to be anchored in the new master's program Digital Humanities at TU Dresden (starting in winter semester 2022/23). In addition, it will be made available as an open educational resource and indexed in relevant directories.

Acknowledgements

We would like to thank Gregor Mitzka, who supports both the current programming course and ExDiMed as a student assistant.

The ExDiMed project is funded by the *Stiftung Innovation in der Hochschullehre* as part of the virTUos network.

Literature

[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/teachlearninqu.10.4

[9] Haaranen, L. Programming as a Performance. Livestreaming 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