Lessons Learned 5, 2 (2025)
Submitted: September 22, 2025
Accepted: January 8, 2026

DOI: https://doi.org/10.25369/ll.v5i2.105
ISSN: 2749-1307

Programming for the Humanities and Social Sciences

Y. Frommherz*, S. Meier-Vieracker, A. J. Matz

Chair of Applied Linguistics, Institute of German Studies and Media Cultures, Faculty of Language, Literature, and Cultural Studies, TU Dresden

Abstract

The ongoing digitization in the humanities and social sciences (*HSS*) enables and requires the use of new, particularly quantitative, methods of analysis. Programming skills give researchers the most flexible way of working with data. However, these are not typically taught in HSS degree programs, and existing teaching/learning materials usually require prior technical knowledge. For this reason, the *virTUos* subproject *ExDiMed* has developed low-threshold, modular self-learning materials for teaching coding skills in the programming language Python, which are specifically tailored to the needs and requirements of HSS students. The content, in the form of interactive Jupyter Notebooks, is available to learners and teachers as OER and has been successfully tested and refined at TU Dresden over six semesters in a flipped classroom seminar. With the advent of generative AI, the course has been expanded in an exploratory manner to sensitize students to the constructive use of AI in coding. Our evaluation shows that teaching basic programming skills has become more relevant in the age of AI, as the latter can only act as a catalyst for coding practice if students already have the basics. Our teaching/learning concept is now firmly established in the *Digital Humanities* master's program at TU Dresden and will continue to be developed after the end of the project.

Die fortschreitende Digitalisierung in den Geistes- und Sozialwissenschaften (*GSW*) ermöglicht und erfordert den Einsatz neuer, insbesondere quantitativer Analysemethoden. Den flexibelsten Umgang mit Daten erhalten Forschende durch Programmierkenntnisse. Typischerweise werden diese aber nicht in GSW-Studiengängen vermittelt und bestehende Lehr-/ Lernmaterialien setzen meist technisches Vorwissen voraus. Deshalb hat das *virTUos*-Teilprojekt *ExDiMed* niedrigschwellige, modular aufgebaute Selbstlernmaterialien zur Vermittlung von Codeskills in der Programmiersprache Python entwickelt, die sich spezifisch an den Bedarfen und Voraussetzungen GSW-Studierender orientieren. Die Inhalte in Form interaktiver Jupyter Notebooks stehen Lernenden wie Lehrenden als OER zur Verfügung und wurden an der TU Dresden während sechs Semestern in einem Flipped Classroom-Seminar erfolgreich erprobt und verfeinert. Mit dem Aufkommen generativer KI wurde das Angebot explorativ erweitert, um Studierende für konstruktiven KI-Einsatz beim Coden zu sensibilisieren. Unsere Evaluation zeigt, dass die Vermittlung grundlegender Programmierkenntnisse in Zeiten von KI an Relevanz gewonnen hat, da letztere nur dann als Katalysator der Codepraxis wirken kann, wenn Studierende bereits über Basics verfügen. Unser Lehr-/Lernkonzept ist mittlerweile im Masterstudiengang *Digital Humanities* an der TU Dresden verankert und wird auch nach Projektende weiterentwickelt.

*Corresponding author: yannick.frommherz@tu-dresden.de
This article was originally submitted in German.

1. Introduction

The humanities and social sciences (*HSS*) have undergone a digital turn in recent years [1], [2], [3], [4], which is likely to be further accelerated by recent leaps in the development of generative AI [5], [6], [7]. Scientists in these disciplines now have access to large amounts of digital data (e.g., social media corpora or OCR data), which both enable and require the use of quantitative analysis methods. Moving beyond low-threshold tools that allow specific, predefined evaluations (e.g., corpus analysis software such as *SketchEngine*), programming skills enable HSS researchers to flexibly access and preprocess large quantities of data that are relevant to their research, analyze it, and prepare it for dissemination [8].

However, given the traditionally qualitative focus of many HSS programs, quantitative methods in general and programming skills in particular are not typically part of the corresponding curricula. Against this background, the subproject *Experimentierraum Digitale Medienkompetenz* (*ExDiMed*) of the *virTUos* consortium in 2021 set out to develop a target-group-oriented teaching/learning format that imparts precisely these skills to HSS students, enabling them to fully exploit the potential of the digital turn [8].

Specifically, during the project period, we developed and tested materials that teach programming skills in Python as a fundamental digital media competence. As the de facto scientific standard, this relatively easy-to-learn programming language offers the possibility of solving research questions and application problems from the HSS in a needs-oriented manner [8], [9].

This article reflects on the work carried out within *ExDiMed*, beginning with the specific needs and requirements of HSS students, followed by our self-learning materials based on latter, their testing in a flipped classroom seminar, and the integration of generative AI, which has emerged in the meantime. We conclude with an outlook. Throughout the project, iterative feedback from learners on our approach and the specific materials was incorporated into their further development (see diagram of the work process in Fig.1). In line with the title of this journal, we would like to present the most important *lessons learned* from our project.

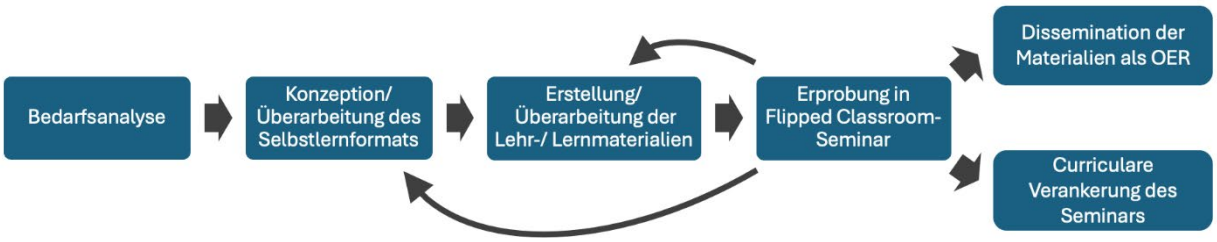


Fig.1 : Schematic representation of the iterative work process.

2. Starting Point

High-quality introductions to programming that are also freely available on the internet are widely available in various formats, from classic textbook-like websites to YouTube tutorials and interactive online courses. Apart from notable exceptions such as *Programming Historian,* however, these are almost exclusively aimed at learners with a certain affinity for technology, mathematics, and logic. Explanations often require relevant prior knowledge, and examples and exercises are mostly numerical in nature (e.g., outputting the Fibonacci sequence). However, our target group, HSS students, generally have no prior knowledge of programming and little technical know-how in general. To the contrary, there are often reservations about computational work. Apart from the importance of content that is as free of prerequisites as possible (see below), the initial focus in developing our teaching/learning concept was therefore on the question of which format would be most effective in imparting programming skills to the given target audience.

3. Open-Source Self-Learning Materials

While our target group is united by the fact that they often have little technical experience and text data is generally very important to them, the various HSS are thematically and method-

ologically diverse. For example, history students have different concepts and interests than those studying linguistics. Beyond the subjects, all students also have individual needs and requirements. In order to take this diversity into account and address as broad an audience as possible within the HSS, we have opted for *self-learning materials* in the format of Jupyter Notebooks, which are also *modular* in structure.

Once students have downloaded our freely accessible content from GitHub, they can work on it flexibly whenever and wherever they want and, crucially, at their own pace. This allows them to repeat individually challenging passages as often as necessary.Feedback obtained from pretesters as well students of the seminar confirmed that learners greatly appreciate this flexibility.

Jupyter Notebooks are a suitable format because they elegantly and unobtrusively interweave code with text blocks (see Fig. 2). The latter contain explanations and formulate tasks and cannot be edited by learners. Code fields, on the other hand, are either empty (e.g., for exercises, see Fig. 3) or already contain code that students can expand ("Complete the code"), modify ("Insert relevant parameters" or "Find the error"), and, in any case, execute. In contrast to traditional textbooks, which require that code be copied exactly, and script-based tutorials, which must first be executed in the command line, learners in our interactive notebooks immediately experience the effects of the given code or specific changes to it. The division of code into several small blocks (with text in between, if necessary) further sensitizes learners to the incremental nature of programming: a problem is solved step by step, and after each block it is immediately apparent whether the code delivers the desired (interim) result. Our teaching/learning format thus combines self-directed knowledge acquisition with immediate application. Embedded graphics and videos, which convey a realistic picture of exploratory trial-and-error coding, further create a multimodal learning experience.

Fig. 2: Excerpt from a Jupyter notebook consisting of text and code blocks that introduce conditional statements. The code blocks are executed.

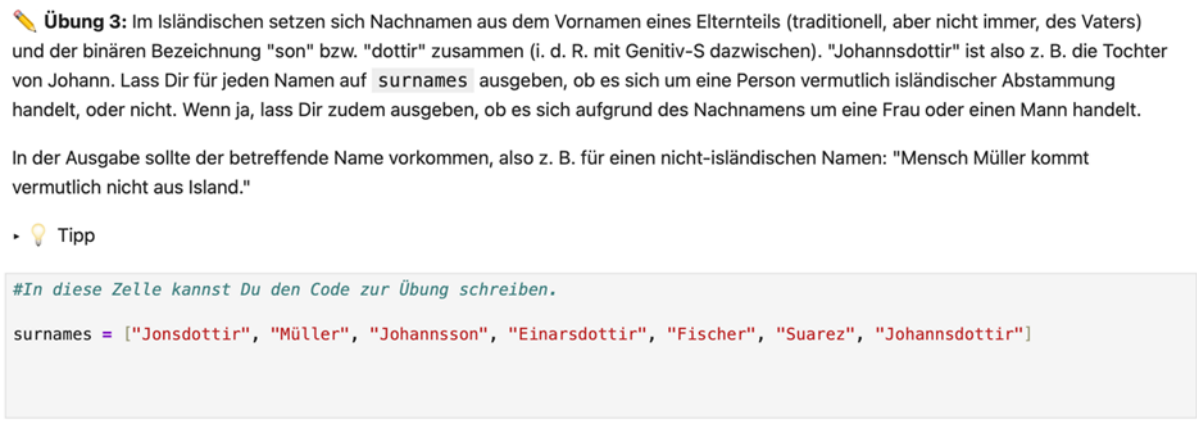


Fig. 3: Example of an exercise on conditional statements, which is thematically derived from the HSS. Optionally, a hint can be displayed that suggests a suitable method (`endswith`).

As mentioned, our notebooks follow a modular structure. We distinguish between a basic module and advanced modules. In the former, students acquire the theoretical and practical foundations of programming with Python, i.e., knowledge of variables, operators, data types, control structures, useful functions and methods, and how to handle external data. With our target group in mind, explanations, examples, and exercises are deliberately taken from the HSS and designed to be practically relevant (e.g., in *Nathan the Wise* the distribution of contributions per character is analyzed , see also Fig. 3). Even concepts that are supposedly generally understood, such as working memory or file paths, are explained explicitly. In addition, we illustrate potentially counterintuitive techniques using metaphors borrowed from everyday life – for example, *indexing* using the idea of numbered houses along a street, where specifically odd house numbers can be "addressed" using the *step* parameter (`muster-strasse[0:20:2]`). Our teaching experience in recent years has shown us that materials for teaching programming skills to HSS students must be designed to be as low-threshold as possible so that learners are not deterred. The fact that our contents are in German, unlike most other introductions to programming, also contributes to their accessibility for our target group.

Beyond the specific content, we ultimately try to get learners interested in programming in three ways. First, right at the beginning, we explicitly motivate the use of programming skills in the HSS by using examples to show what can be achieved in these disciplines with the help of code, which would be very time-consuming or even impossible to implement with a manual approach. Second, we introduce algorithmic thinking before getting into the specifics of Python. This is the ability to 1) analyze a given problem precisely, 2) find expedient steps to solve it, 3) operationalize these steps, i.e., formulate a complete and correct algorithm, 4) test this algorithm on both typical and atypical cases, and 5) improve its efficiency [4, p. 95]. Internalizing this fundamental approach to a (programming) problem seems to be more important than the effective syntax and semantics of the respective programming language [4, p. 90]. Results from an iteratively designed programming course, in which only one parameter was adjusted from the previous course, suggest that even a short unit on algorithmic thinking leads to better programming skills [9]. This is consistent with our experience that learners experience a kind of "aha" moment when they become aware of this completely logical way of thinking. Third, we try to proactively prevent disappointment and dropouts by making it clear to learners at various points in our notebooks that they will often fail, but that this is really quite normal (see Fig. 4)

.

▾ 🔍 **Exkurs: Hilfe holen**

Bis zu diesem Punkt bist Du sicherlich auf das ein oder andere Problem gestoßen, das Du nicht selbst lösen konntest. Vermutlich hast Du in diesen Fällen eine KI konsultiert oder Hilfe bei Google gesucht. Das sind genau die richtigen Strategien! 😄

In den seltensten Fällen bist Du mit Deinen Programmierproblemen alleine, weshalb es im Internet meist bereits viele Lösungsvorschläge dazu gibt. Dies trifft besonders auf Python zu, da die Programmiersprache sehr weit verbreitet ist. Diese Hülle und Fülle an Hilfestellung zu Python im Internet ist denn auch der Grund dafür, warum KIs wie chatGPT oder GitHub Copilot ziemlich zuverlässig Hilfe leisten können. Schließlich basiert ihr Output auf Trainingsdaten, die zu einem großen Teil aus dem Internet stammen.

Lass dir bei Problemen zunächst von KI helfen – in aller Regel ist dies am effizientesten. Überprüf aber stets, ob die vorgeschlagene Lösung Dein Problem wirklich behebt und Du den Code nachvollziehen kannst. Falls nicht, kannst Du Dir den Code von der KI erklären lassen oder eine simplere Version erbitten.

Fig.2 : Excerpt from a notebook on dealing with errors and problems, in which we also engage in implicit expectation management.

Building on the basic notebooks, additional self-contained notebooks introduce data analysis with `pandas`, regular expressions, web scraping, and tagging (e.g., lemmatization or sentiment analysis). This allows learners from the various HSS to put together a modular programming course that is relevant to them. Suppose a student wants to enrich an existing language corpus with additional information such as word types or syntactic dependencies. In that case, she can specifically expand her Python skills in the area of tagging. Her fellow student, who wants to compile his very own data set from the internet, can complete his skill set with a focus on web scraping.

In addition to the actual notebooks, we offer extensive additional content. Although the teaching notebooks already contain numerous exercises (see Fig. 3), we provide additional ones (and solutions, of course) for each topic, as we have found that students learn best through independent, creative application of theory. There are also materials on topics such as using the command line, Git(Hub), data visualization, and training your own basic language model.

As mentioned, all materials are stored in a separate GitHub repository and referenced as Open Educational Resources (*OER*) under a CC-BY-SA license on the platforms twillo and Wikiversity. They are also available to teachers – thanks to their modular structure, our materials can be used as-is or flexibly supplemented with additional content.

4. Testing in Flipped Classroom Seminars

Since the 2022/23 winter semester, the materials described above have been used in a seminar regularly offered by the Chair of Applied Linguistics. In the spirit of exchange between the individual *virTUos* subprojects, we also held a one-time joint course with *DigitalHerrnhut*, in which we prepared and analyzed Herrnhut texts for linguistic-historical research using the simultaneously acquired programming skills.

Our seminar is designed as a flipped classroom format in the spirit of blended learning [10], [11], [12]. Asynchronous and synchronous learning phases are didactically interlinked, with students working on their notebooks primarily asynchronously – flexibly in terms of time, place, and pace – and the synchronous, spatially co-present sessions serving for recapitulation and *joint* practice and coding (see example seminar schedule in Fig. 5). Collaborative learning settings have been shown in various studies to be conducive to teaching programming skills [1], [9], [13]. In our seminar, the lecturer solves more complex and particularly illustrative programming problems in a frontal manner, but always with the involvement of the students. Here, too, the incremental and iterative practice of coding becomes clear, while at the same time care is taken to cultivate an error-tolerant mindset.

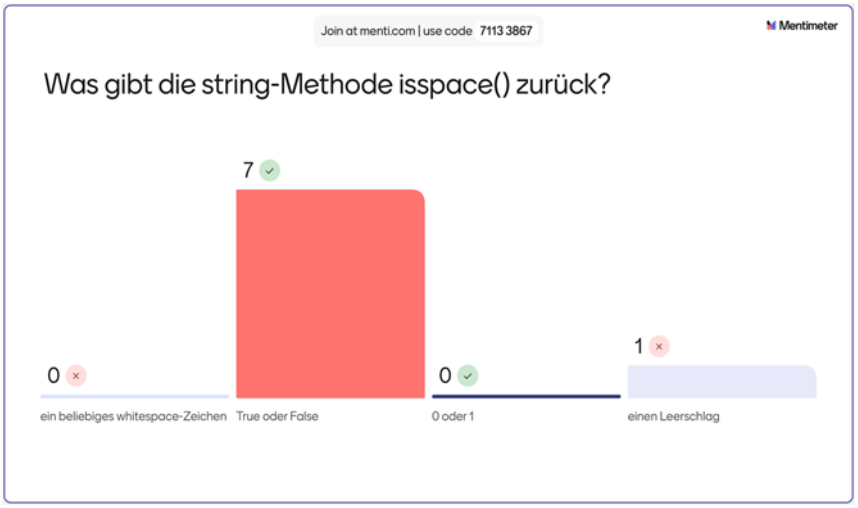
Datum	Zeit	Synchron	Asynchron (nach Sitzung)
16.10.	13:00-14:30	Installation & Algorithmisches Denken	„Einführung“
23.10.	13:00-14:30	„Einführung“ rekapitulieren	„Datentypen“
30.10.	13:00-14:30	„Datentypen“ rekapitulieren	„Kontrollstrukturen“
06.11.	13:00-14:30	„Kontrollstrukturen“ rekapitulieren	„Funktionen und Methoden Teil 1“
13.11.	13:00-14:30	„Funktionen und Methoden Teil 1“ rekapitulieren	„Funktionen und Methoden Teil 2“
27.11.	13:00-14:30	„Funktionen und Methoden Teil 2“ rekapitulieren	Vorbereitung Programmieren mit KI
04.12.	13:00-14:30	Programmieren mit KI	„Input und Output Teil 1“
11.12.	13:00-14:30	„Input und Output Teil 1“ rekapitulieren	„Input und Output Teil 2“
18.12	13:00-14:30	„Input und Output Teil 2“ rekapitulieren	„Datenanalyse Teil 1“
08.01.	13:00-14:30	„Datenanalyse Teil 1“ rekapitulieren	„Datenanalyse Teil 2“
15.01.	13:00-14:30	„Datenanalyse Teil 2“ rekapitulieren	*„Reguläre Ausdrücke“*
22.01.	13:00-14:30	Vorbereitung Mini-Hackathon	*„Tagging“*
29.01.	13:00-17:00	Mini-Hackathon	Vorbereitung Projektarbeit
05.02.	13:00-14:30	Vorbereitung Projektarbeit	

Fig.3 . Sample seminar schedule from the 2024/25 winter semester, showing the integration of asynchronous and synchronous learning phases in line with the flipped classroom concept. This seminar was also used as a test bed for the integration of generative AI (see below).

On the other hand, we implement settings that promote collaboration among students. At the beginning of each semester, we create peer groups consisting of two or three students – with different levels of prior knowledge, if possible – who are to work together in synchronous sessions throughout the seminar. The highlight of collaboration among students is a *mini-hackathon* at the end of the semester. For four hours, participants devote themselves to a data set they have chosen themselves, develop questions that interest them, and set about operationalizing these questions with the help of code. At the end of the mini-hackathon, they present their results and reflect on the process. Such mini-hackathons enable students to co-construct ideas and solutions that they might not have developed on their own [1]. Over the course of the semester, various corpora were analyzed, such as stenographic Bundestag minutes, user comments on the Instagram project *@ichbinsophiescholl* [14], soccer game statistics, and newspaper articles. The first dataset, for example, was analyzed with regard to the distribution of interjections in parliament by political groups and individual members of parliament. Although students can apply their growing Python skills throughout the semester in thematically isolated exercises, the mini-hackathon is the first time they are used on a larger and more holistic scale. Most importantly, however, coding is guided by the interests of the students themselves, which has a positive effect on their motivation. In addition to these collaborative components of the seminar, gamified aspects such as knowledge quizzes are particularly popular, as they offer a break during the session and seem to be appealing due to their competitive nature (see Fig. 6). The newly acquired programming skills are ultimately applied in the final assignments following the seminar. Typically, students complete their own small research project for this purpose – from data collection (e.g., using web scraping) to data preparation (e.g., using regular expressions) and evaluation (e.g., with `pandas`) to visualization of the results (e.g., with `matplotlib`). Many students report that their projects have enabled them to make another big leap in their skills, partly because they have now turned to more complex code, driven by their own individual interest in knowledge. Over the course of the semester, many high-quality exam papers were submitted, including a linguistic paper on Helvetisms and Austriacisms in German-language newspaper articles, a historical paper on mapping the Digital Herrnhut cemetery, and an visual studies paper comparing historical mass production strategies with digital reproduction methods.

The flipped classroom format of our seminar, which is fundamentally derived from our overarching self-learning concept (see above), has proven itself over the last six semesters. Although some students were occasionally overwhelmed by having to work through most of the learning content on their own, the majority of participants appreciated the format or at least pragmatically recognized that the only way to really learn programming is by coding, coding, and coding again—and that simply has to be done *independently*. Apart from the fact that weekly sessions would not be sufficient for this, neither the teacher nor collaborative exchange with peers can substitute this for learners. These elements, which are anchored in the synchronous parts of the seminar, offer important support for the individual learning process, but cannot replace it. At the same time, the past few years have shown that synchronous sessions on more complex topics towards the end of the seminar should definitely last longer than two hours, if only because the exercises are much more time-consuming at this stage compared those covering introductory content. A complete block seminar that bundles all synchronous parts into a few full-day units has not proven successful, however, as students' mental capacity is limited and the independent programming practice that is so central to the course seems to be less established than in a course that spans the entire semester.

The seminar described above is now part of the curriculum of the new Master's program in *Digital Humanities* at TU Dresden, but students from other disciplines, including various teacher training programs, also regularly enroll. The seminar is continuously being improved, not least with regard to generative AI. Its emergence in the middle of the project raised new questions and necessitated adjustments, which are described below.



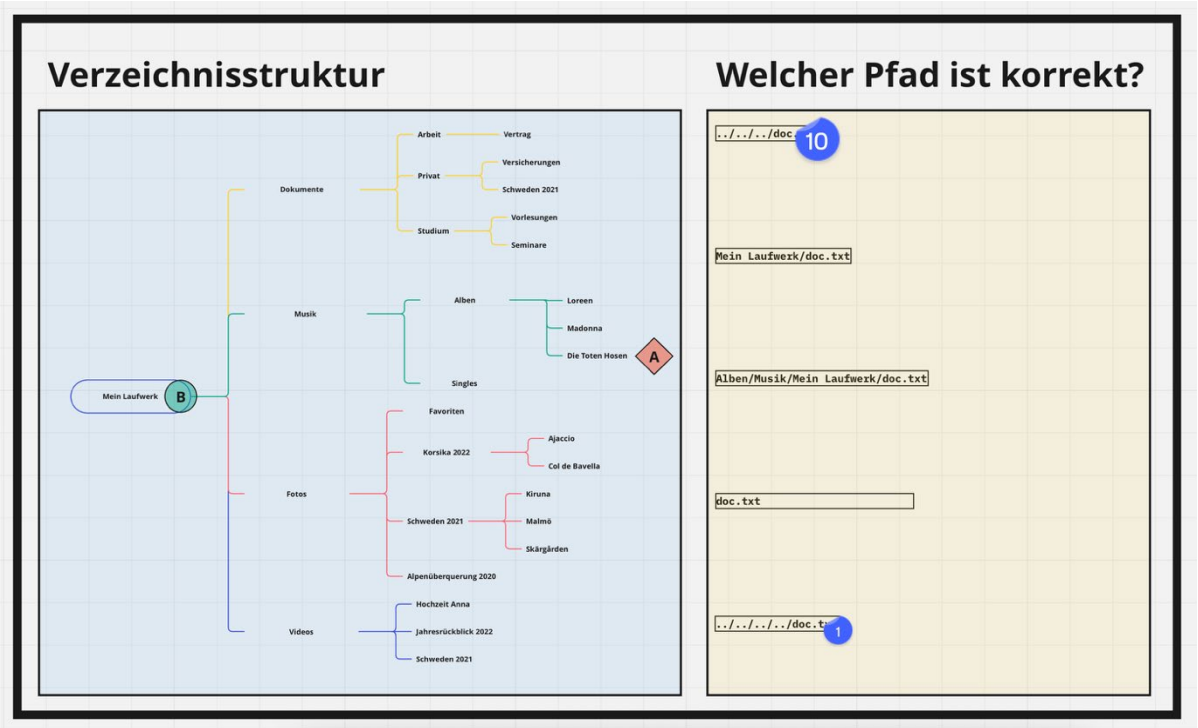


Fig.4 : Gamified seminar components: knowledge quiz on string methods (top; implemented with mentimeter.com) and on relative file paths for navigating from a given directory A to directory B (bottom; miro.com).

5. Generative AI

Current generative AI tools such as *chatGPT* or *Google Gemini* are now capable of generating programming code fully automatically and mostly error-free [15]. Once this became apparent, we were faced with the fundamental question of whether learning to program was still worthwhile for HSS students. For three reasons, we were fundamentally convinced that programming skills in the disciplines in question are not superfluous in the face of powerful AI, but on the contrary, may be more relevant than ever. First, experience to date has shown that our target group rarely considers code – even AI-generated code – as a problem-solving strategy due to the technical inhibitions described above and generally low digital literacy. Against the backdrop of the digital turn in the HSS, a low-threshold introduction is therefore likely to remain essential in order to familiarize such students with the analysis methods that have become relevant [8]. Second, generative AI can only be used if meaningful prompts are formulated. However, programming-related prompt engineering requires an understanding of when code is useful, what code can (and cannot) do, and

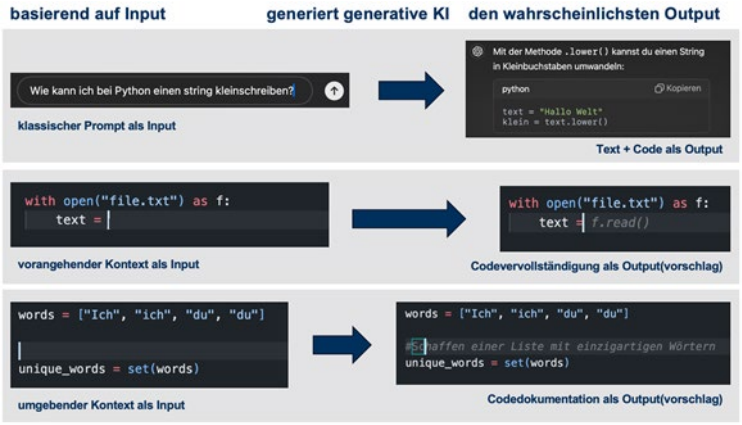


Fig.5 : Theoretical input on how generative AI works.

knowledge of the relevant terminology. Learners need these same skills to be able to evaluate generated output and take action in the event of faulty AI code. These skills too need to be taught in a way that is appropriate for the target group. Thirdly, generated code must always be executed. Some AI tools allow code to be executed in virtual environments. However, once local files are used, which will soon become unavoidable when programming for research purposes, a separate *functioning* development environment (*IDE*) is required, for which the generated code usually has to be adapted, e.g., with regard to file paths. In our experience, however, setting up such an IDE represents a major entry barrier that needs to be overcome through appropriate learning offerings.

If programming skills in the HSS and their teaching in higher education have not become obsolete as a result of generative AI, the next question is how a teaching/learning format such as ours needs to be adapted to the new conditions that AI creates for programming. In order to explore the meaningful integration of AI, we used our seminar in the winter semester 2024/25 as a test bed.

A survey at the beginning of the seminar revealed that students used generative AI in many ways, but not for programming. As we anticipated, the reasons given were a lack of basic knowledge about prompting and evaluating output, as well as a fear of errors in the AI code. Since we assumed that certain basic knowledge must be in place before AI can be used profitably for programming, we scheduled a corresponding introduction for the middle of the semester (see seminar plan of the testbed seminar in Fig. 5). We first used theoretical input to ensure that all participants had the same level of understanding of how generative AI works: Based on input – whether in the form of a classic prompt or code context – the AI suggests the most likely output (see Fig. 7). We then demonstrated the probabilistic basis of generative human and artificial intelligence using natural language cloze texts and incomplete code, which the students were asked to complete individually. Unsurprisingly, depending on the "input," certain "outputs" were more likely than others, just as is the case with generative AI "behind the scenes" (see Fig. 8).

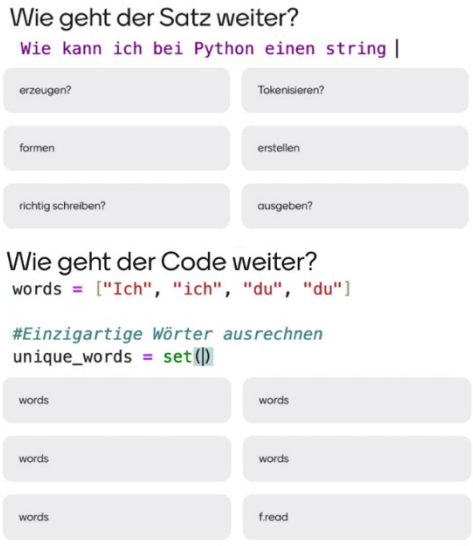


Fig.6 : Raising awareness of the probabilistic basis of generative AI. Students were asked to complete the sentence/code individually. The screenshot shows some of their answers.

We then differentiated between four use cases in which generative AI can be used constructively: code generation "from scratch," debugging, commenting or documenting code, and having code explained. Each type was then practiced using examples and employing both "general" AI such as *chatGPT* and AI optimized for coding, *GitHub Copilot*. As the seminar progressed, as well as for the mini-hackathon and final assignments, students were free to decide whether and how to use generative AI. The instructor continuously demonstrated its use in synchronous sessions.

After the seminar, the tested AI integration was evaluated by externally moderated focus groups to gather practical experiences from the students [5]. According to this, students rated both the type and timing of the introduction in the middle of the semester as very useful and also appreciated the self-determination in the use of the tool. Most students reported that they would never have started programming without our low-threshold introduction, but now felt capable of using AI competently. The primary areas of application for AI turned out to be the correction of syntactic errors and code documentation. Students also like to have code "broken down". i.e., explained. At the same time, it was reported that completely AI-generated code is often too complex and creates new problems, e.g., by way of dependencies. Not least because of this, interacting with AI sometimes takes longer than finding a solution on one's own. Diverging attitudes toward AI use in peer groups and individual inhibitions (self expectations; fear of making mistakes; fear of losing one's own creativity; the feeling of cheating) also seem to stand in the way of (increased) use of generative AI in programming. Another interesting finding was the recurrent statement that students preferred to ask the lecturer rather than the AI, as the former reacted in a more human way by embedding his answers in the real seminar context.

It is undisputed that code remains a central scientific problem-solving strategy, also in the increasingly digital HSS. Generative AI proved to be a helpful catalyst for learning and applying programming skills in our test seminar. However, as we assumed, it became clear that basic knowledge must first be available in order for learners to benefit from generative AI in their programming practice. This confirmed that low-threshold, target group-sensitive teaching of programming skills in higher education is very relevant, especially in times of powerful AI. Only in this way can the potential of the digital turn described at the beginning be effectively exploited. Accordingly, the tested form of AI integration into the seminar was retained in the 2025 summer semester. The teaching/learning materials were also revised to point out the possibility of AI use at appropriate points (see also Fig. 4).

6. Outlook

By way of our contents and seminars, *ExDiMed*'s goal was to enable learners from various HSS programs with no prior knowledge to address issues relevant to them in a digitally competent manner – from obtaining their own data to data preprocessing, evaluation, and visualization. Numerous good final assignments, as well as, e.g., programming-intensive master's theses, and feedback from learners over six semesters, show us that we have come a long way toward achieving this goal, at least for the HSS at TU Dresden.

While both the content creation and the seminar have been largely carried out by one single research assistant, a handover to new teaching staff is currently taking place, who will continue developing both the materials and the course after the end of the project. To this end, all processes, such as the handling of the GitHub repository, are currently being documented. In addition to creating new advanced modules, the focus in the future will be on the ongoing onboarding of new developments in generative AI. Beyond TU Dresden, we hope that our materials will be used by lecturers from the HSS at other German-speaking universities in their teaching, either in their entirety to teach the basics for subsequent subject-specific content, or by using individual advanced modules to supplement an existing course.

Acknowledgements

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

Literature

[1] L. Beck und A. Chizhik, „Cooperative learning instructional methods for CS1: Design, implementation, and evaluation“, ACM Trans. Comput. Educ., Bd. 13, Nr. 3, S. 10:1-10:21, Aug. 2013, doi: 10.1145/2492686.

[2] H. Snee, C. Hine, Y. Morey, S. Roberts, und H. Watson, „Digital Methods as Mainstream Methodology: An Introduction“, in Digital Methods for Social Science: An Interdisciplinary Guide to Research Innovation, H. Snee, C. Hine, Y. Morey, S. Roberts, und H. Watson, Hrsg., London: Palgrave Macmillan UK, 2016, S. 1–11. doi: 10.1057/9781137453662_1.

[3] Deutsche Forschungsgemeinschaft, „Digitaler Wandel in den Wissenschaften“, Okt. 2020, doi: 10.5281/ZENODO.4191345.

[4] F. Jannidis, H. Kohle, und M. Rehbein, Hrsg., Digital Humanities. Eine Einführung. Stuttgart: Metzler, 2017.

[5] Y. Frommherz, A. J. Matz, und S. Meier-Vieracker, „KI-gestütztes Programmierenlernen – Erprobung und Erfahrungen eines Lehr-Lernkonzepts“. Zugegriffen: 14. August 2025. [Online]. Verfügbar unter: https://fis.uni-bamberg.de/handle/uniba/107576

[6] G. Punziano, „Adaptive Epistemology: Embracing Generative AI as a Paradigm Shift in Social Science“, Societies, Bd. 15, Nr. 7, S. 205, Juli 2025, doi: 10.3390/soc15070205.

[7] S. Schiller-Stoff, L. E. Münzer, C. Dittmann, und S. Sagadin, „Der Einfluss von AI-Pair-Programmers auf die Digital Humanities: Potentiale und Limitationen“, in Book of Abstracts DHd 2025, Bielefeld, 2025. doi: https://doi.org/10.5281/zenodo.14887461.

[8] Y. Frommherz und J. Langenhorst, „Digitale Kompetenzen für Geistes- und Sozialwissenschaftler:innen. Vorzüge eines Blended Learning-Formats für die Vermittlung von Programmierkenntnissen.“, Lessons Learn., Bd. 2, Nr. 1, Juli 2022, doi: 10.25369/ll.v2i1.37.

[9] T. Koulouri, S. Lauria, und R. D. Macredie, „Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches“, ACM Trans. Comput. Educ., Bd. 14, Nr. 4, S. 1–28, Feb. 2015, doi: 10.1145/2662412.

[10] S. Seufert und P. Mayr, Fachlexikon e-lerning: Wegweiser durch das e-Vokabular. in managerSeminare. Bonn: ManagerSeminare May, 2002.

[11] E. Schoop, H. Bukvova, und C. Lieske, „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, Riga, Lat., Dresden: TU Dresden, 2010. [Online]. Verfügbar unter: https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa-26183

[12] M. Kerres, Mediendidaktik: Konzeption und Entwicklung mediengestützter Lernangebote. München: OLDENBOURG WISSENSCHAFTSVERLAG, 2013. doi: 10.1524/9783486736038.

[13] G. Braught, T. Wahls, und L. M. Eby, „The Case for Pair Programming in the Computer Science Classroom“, ACM Trans. Comput. Educ., Bd. 11, Nr. 1, S. 2:1-2:21, Feb. 2011, doi: 10.1145/1921607.1921609.

[14] S. Meier-Vieracker, „‚LIEBE SOPHIE‘ – ADRESSIERUNG UND INVOLVIERUNG IN INSTAGRAM-KOMMENTAREN AM BEISPIEL DES PROJEKTES @ICHBINSOPHIE-SCHOLL“, Dez. 2023, doi: 10.48694/KORDAF.3849.

[15] A. Ziegler u. a., „Productivity assessment of neural code completion“, in Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, San Diego CA USA: ACM, Juni 2022, S. 21–29. doi: 10.1145/3520312.3534864.