Teaching Computational Methods to Humanities Students

Emily Öhman[0000-0003-1363-7361]
University of Helsinki
emily.ohman@helsinki.fi

February 10, 2019

Abstract

This paper discusses the academic and societal implications of teaching computational methods to humanities students from the perspective of digital humanities. Pedagogical choices are backed up by both pedagogical theory and concrete examples from actual courses and course feedback. The aim of this paper is to introduce clear best-practice recommendations for developing digital humanities teaching with an emphasis on methods teaching in order to increase the number of students who understand such methods and can apply them to their own projects.

1 Introduction

As early as 1991, Christian Koch [26], professor of computer science at Oberlin College, wrote about the dangers of bringing in a computer scientist (technical person) to help with computational or technical tasks in teaching or research as it "preserves and even underscores a division of labor between an executive or idea type person and a secondary implementer or technical person and it undercuts the idea of a unified approach to a field of study" [26]. Instead, he argued that humanities scholars and teachers should learn computational methods and algorithmic thinking.

The importance of developing these skills on an institutional level is something that has been emphasized by many for more than 35 years already [21, 26, 30], and it seems that there is finally a more widespread understanding of the importance of these methods for the humanities based on the increased visibility and funding of digital humanities projects [1, 2, 3]. However, the adoption rate of computational skills by humanities scholars is still low in practice [5, 9, 20]. The only way to combat this is to teach all humanities students basic computational literacy starting at undergraduate level.

Too often, humanities students realize at graduate-level that they need computational skills for their research, at which point it is a scramble to try to learn algorithmic thinking and code-literacy so that they can conduct the necessary steps to be able to finish their Master's thesis or even a PhD dissertation.

The focus of this paper is a very large (150+ students) online course for undergraduate students for which we have implemented a blended learning approach. The course is known as "Introduction to Language Technology", and until last year (2017) it was compulsory to all of those minoring or majoring in linguistics or language technology. Currently it is compulsory to those still under the old degree system, and those minoring or majoring in language technology. Regardless, the number of students on the course has increased. For many students it is their first university course ever, and for most other students it their first computer science-related course. Another course, "*Methods for Digital Humanities*[1]" will also be mentioned. This course is a smaller (roughly 20 students) in-class course aimed at graduate students. What both of these courses have in common is the teaching of computational methods to students with highly variable computational skills where sometimes technophobia is the highest hurdle to overcome.

The aim is to present some best practices for any teacher dealing with teaching computational methods to students without a strong computational background. This is done through examining the course contents, teaching environment, and by evaluating the results of formative and summative assessments. The use of blended learning and technological tools in teaching specific concepts is reviewed, and suggestions on how to deal with technophobia and reluctant learners are provided.

The next sections will discuss the theoretical background of digital humanities pedagogy, and blended learning (section 2), as well as how the courses were implemented and how they were improved based on student feedback (section 3). The paper ends with a concluding discussion on best practices (section 4).

2 Previous Work

There is a growing inequality in society between those who are able to program and those who are not [13, 31]. Rushkoff [31] has even gone so far as to state that the choice is "program or be programmed". If this is true in society, it is even more so in academic research.

Today many digital humanities projects are multidisciplinary collaborations between humanities scholars and computer scientists. For digital humanities to survive as a field, there needs to be a shift towards interdisciplinary work. In the long run it is not enough for a humanities scholar to collaborate with a computer scientist to generate structured data or for a computer scientist to have a humanities scholar interpret the data he/she has created (i.e. multidisciplinary work). Furthermore, it has been shown that often in such collaborations it is the computer scientist who does the interpretation of the data as well as the computational part [7]. Digital humanities scholars need to be able to do both things (i.e. interdisciplinary work) to be able to even do the interpretation [7, 16, 26, 29, 35], and the creation of a truly interdisciplinary framework starts at undergraduate level teaching.

[1] This course was designed and implemented by University of Helsinki Assistant Professor of Humanities-Computing Interaction Eetu Mäkelä: eetu.makela@helsinki.fi.

2.1 Teaching Digital Humanities

There has been a long-standing difference in opinion in digital humanities pedagogy about whether it is useful to teach students how to use ready-made software (such as *AntConc* [6], *Voyant Tools* [33], and even *DH Box* [23]) without the students having first gained an understanding of what is going on ”behind the scenes”. On the one hand, some aspects of computational methods have a rather steep learning curve that require specialized knowledge. This may be discouraging to some students and might even deter them from pursuing digital humanities altogether [23].

However, despite the popularity of such ”black boxed” tools that ”hide computation”, without an understanding of computational methods and basic statistics, there is a very real risk of skewed data resulting in meaningless output possibly without anyone even noticing [16, 37]. An example would be blindly using several text files (books) from Project Gutenberg [4] with some black boxed tool to count frequencies or collocations while being unaware of the metadata present in all files from Project Gutenberg. The output in this case would not reflect the true nature of the actual contents of the text files. This does not mean that digital humanities scholars need to become programmers, but they should understand the basics of coding and be able to read code even if they are not proficient in writing it themselves. It is imperative to understand, not only the structure of the texts one is investigating, but to understand the structure of texts in general as they pertain to machine readability.

It should be noted that research computing has also undergone a shift corresponding to the increase in digital materials and methods. The focus is no longer on running infrastructure for technical projects in science disciplines, but instead nearly all projects in vastly different fields have some research computing needs. The PIs, however, have highly variable technical expertise [11]. In practice this means that a non-technical humanities scholar needs to be computer literate enough to at least be able to convey what it is they need and understand the limitations of computing too, even if they are not doing any of the computing.

Given all this, the education of digital humanities scholars needs to start at undergraduate level at the latest. Naturally, a strong knowledge of one’s own field in the humanities is still of utmost importance, but there is no reason to think that the teaching of computational methods throughout an undergraduate degree would lead to a decreased understanding of the underlying principles and philosophies of the humanities. Broader integration and embedding of digital humanities skills development into undergraduate curricula is now more than ever becoming a matter of urgency [14]. Any undergraduate program in the humanities should include repeated and diverse courses in both method- and topic-intensive courses [35]. We have managed to implement this line of thinking at the University of Helsinki, however, more courses that build upon each other and are compulsory to all humanities students still need to be developed.

2.2 Teaching Coding

”For some children coding was a magical process that required supernatural skills” writes Dufva [13] about creative coding classes for children. Although this quote was regarding children’s attitudes towards programming before they learned to code, it is

descriptive of many humanities students too. Programming is now part of the national curriculum in Finland, however, this is a recent development and most university students, particularly in the humanities, have no coding experience. The view of code, and by extension computers, as something almost supernatural is prevalent among students and seems to be one of the main causes of technophobia. Likewise, not only students of the humanities but teachers, too, often lack programming skills. This can lead to an unnecessary fear of incompetence as described by one of the aforementioned children's programming teachers [13].

It may seem that I am advocating for all humanities students and teachers to immediately enroll into programming classes, but that has proven to generally not be a very efficient way to teach humanities students how to code as the courses offered by most computer science departments offer few text examples, rather focusing on math, and generally the programming language chosen is not ideal for practical applications in the humanities [15, 27, 29]. In our courses we mainly use *Python*. This is because *Python* does not have a steep learning curve, it is easy to create short stand-alone scripts with it, and plenty of material for processing natural language with *Python* exists (*NLTK*, *word2vec*, etc.). It has been shown that beginners make fewer mistakes when they learn Python first compared to traditional programming languages such as Java or C++ [18]. Furthermore, as there is a strong language technology component in digital humanities at the University of Helsinki, and *Python* is often the language of choice for language technology, the existing departmental knowledge of *Python* made it an easy choice despite some other departments favoring *R*, and Ramsay favoring *Ruby* [29].

2.3 Blended Learning

Blended learning is the merging of the traditional classroom-based lectures and online teaching [17]. It is a form of teaching that works well for digital humanities courses because it allows for the combination of methods and topics teaching in a seamless manner. Kettula-Konttas and Myyry [24] ran two versions of the same course for several years: one purely lecture-based, and one fully online. They found that for the lecture-based course, they were unable to go into as much depth as they would have liked, but for the online course, the student work was unsatisfactory and did not meet the requirements. They felt that both student learning outcomes and satisfaction greatly increased once the two parallel courses were finally merged into one blended learning course. By doing this, they claim the course much better fit into *Vygotsky's Zone of Proximal Development* [8, 38], i.e. the thought that students, when faced with a new topic, need more "scaffolding" in the beginning to be able to learn more effectively and independently in later stages [24]. The teachers' narrative on the development of their blended learning course is very similar to mine, and gives hope that it is indeed possible to improve the merger of lectures and online environments to as close to an optimal blend as possible.

3 The Course

The course used as an example in this paper is *Introduction to Language Technology (undergraduate, 5 ECTS)*. It is a large online course with up to 150 students. It is possible to complete the course without ever setting foot in a classroom, but many students opt to attend the voluntary computer lab sessions. The computer lab sessions are support sessions designed to help the students complete the assignments. The course has run in one form or another for decades, but four years ago (2015) the content and structure of the course were completely overhauled and modernized to better suit the needs of the students.

The intended learning outcomes for this course are for the students to demonstrate an understanding of the essence of language as it relates to language technology. After the course they should be able to recognize the benefits of and the potential presented by language technology in today's applications as well as future applications, and understand the complexity and requirements of natural language processing. Furthermore, after the course they should be able to apply appropiate practical methods that enable them to use language technology in their own studies and research. These practical skills range from *grep* and regular expressions to corpus linguistic tools such as *AntConc*, and editing snippets of code and directly interacting with a *Python* interpreter.

3.1 Formative & Summative Assessment and Course Feedback

Although there is some variation in how the terms *formative* and *summative* assessment or evaluation are used, in general a summative assessment is the typical end-of-course or unit assessment of students' learning of the course material in order to be able to assign grades describing the level of understanding a student has demonstrated. In contrast, formative assessment is a dialogue between students and teachers in order to help students reach the desired learning outcomes [39].

Traditionally most courses end in some type of summative assessment of students and their abilities related to course content [19, 34, 39]. I have found that the best way to evaluate a practical method-oriented course in computational methods for humanities students is continuous formative assessments where all assignments are contributing to the learning process rather than testing what a student knows. Ramsay takes this even further in his elementary digital humanities course: "I tell students at the beginning of the semester, this class has no papers, no presentations, no quizzes, no midterm, and no final exam....instead problem sets." [29].

Furthermore, we have been actively asking for formative feedback and used this feedback to change assignments on the fly. This has been done via the use of polls, especially via the Presemo platform[2], which allows for anonymous feedback. We have used this to elicit student impressions of how much they are learning each week, and how much they have cumulatively learned during the course in relation to their previous knowledge on the subject. Although all feedback is valuable and even necessary in most cases to fulfill the requirements of constructive alignment, this formative feedback

[2] https://github.com/HIIT/presemo

has mainly confirmed that the problems listed at the end of the course by the students are felt throughout the course.

Figure 1 shows the actual results of a poll that asked students to indicate, on a sliding scale from 0-100, how much of the course topics they were familiar with from before the course (y-axis), and how much of the course content they felt that they had understood after week 3[3] (x-axis). Each dot represents one student.

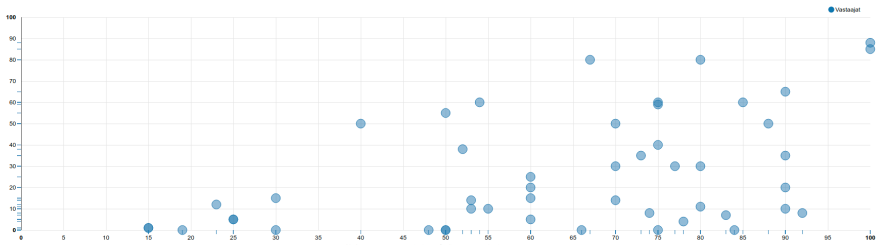

Figure 1: Presemo results: Familiarity (y-axis) and understanding of course topics (x-axis) for *Introduction to Language Technology.*

Although it is clear from figure 1 that those students who had the least knowledge of the course topics from before the course are over-represented among those who have not understood the course topics very well, in reality the same students have managed to successfully complete the course assignments and go on to write interesting final assignments using the methods taught on the course. It is hard to say why people ”vote” the way they do in polls like this, however, I suspect the students who rate their understanding of course topics so far as very low, are those who are the most technophobic. As can be seen, a majority of the students who also had no previous knowledge of the course topics have satisfactorily understood the course topics. There is much variation, too, in how those who rate themselves as having quite a bit of previous knowledge rate their understanding of course topics. It would be interesting to ask students why they vote the way they do; it might even be an avenue for further study into how presumably technophobic students view their abilities.

3.2 Student Feedback

Students have been active during the course with their concerns, complaints, and questions. This has helped us be more aware of which assignments are considered difficult or simply unclear. Here are some excerpts of the feedback received after the course in the fall of 2016:

“It was good that the right answers to the exercises were posted after handing the exercises in. Although the topic of the course was challenging, the teachers tried to be reasonable and helpful.”

[3]Week 3 is about halfway through the course disregarding the time reserved for the final assignment

"Lähiopetuskerrat auttoivat omaa oppimistani merkittävästi verrattuna viime vuoteen, jolloin kurssi jäi itseltäni kesken, kiitos siitä."
Translation: "The lab sessions helped my own learning significantly compared to last year, when I dropped the course, thanks for that."

"I felt that the course in general was very challenging. The topics were completely new to me, so there was a lot of information to process starting from the basics. The exercises were at times difficult, too, but it was rewarding when I got them right...

"Vaikka kurssi on minulle pakollinen, koin olevani hieman ulkopuolella kurssin kohderyhmästä. En tiedä, kuinka koodaamista jo osaavat kokivat kurssin haastavuuden, mutta minulle tuli olo, ettei ei-koodaajataustaisille selitetty kaikkia asioita tarpeeksi alusta asti, eikä kysymyksiäni aina ihan ymmärretty... "
Translation: "Even though the course was compulsory for me, I felt like I wasn't part of the course target audience. I'm not sure how challenging those who already knew how to program found the course, but I felt like those without a programming background did not receive adequate explanations from the start and my questions were often not understood"
NB! This was before *Python* was taught on the course, so the student is likely referring to regular expressions.

"I'm only a freshman so I didn't have a lot of experience but fortunately that was not a problem."

"There were minor issues with the final assignment. Sometimes many of us had problems understanding what you wanted us to do. But in the end I don't think there really was anything unnecessary or bad in the course."

It is clear from the feedback that some students found the course challenging, and that some brought up legitimate concerns which were amended for the next year. Most students, however, seem to still consider the course a positive addition to their skill-set and were happy to suggest ways to improve the course further. Much of the feedback mentioned issues that we had already planned to implement in 2017. These implementations were introductory lectures, more lab sessions, and more instructional videos made by us.

3.3 Feedback-based Revisions of the Course

It is important to develop the course in such a way that a maximum number of students are able to benefit from it. The feedback students supply at the end of the course is generally very positive, and the overall score the students give the course after completion is consistently quite high at around 4 out 5 each year. However, the same complaints remain every year; approximately 25% of students find the course too easy, 25% find it too difficult, and the rest find it just about right. The challenge is to develop the course so that both minority groups will feel like the course is on an appropriate level to their skill-sets.

Several steps have been taken to improve the course along the way. Among these steps were first and foremost an overhaul of the course content to be more contemporary and thus relevant to the needs of current students. Furthermore, the course needed to be more practical to better prepare them for further studies in the fields of language technology and linguistics, and of course to future proof our students.

With the more demanding content step-by-step instructions on how to complete some of the more technical assignments were included. We also added lab sessions for those students that felt they needed extra help. When the assignment deadline has passed, detailed answers are posted online, often in the format of YouTube screen capture videos that clearly show how the task could have been accomplished, why certain things work and others do not etc.

3.3.1 Redesigns

In the first year that I worked on this course (2014 as a course assistant), it was an email-based course. Students would be sent a few pages of reading per week and they were expected to answer 1-3 related questions with a few sentences. The topics were classic language technology, with very few modern articles as course material. The course succeeded in giving the students a cursory overview of the field of language technology, but provided no practical skills whatsoever. The assignments were too easy, this was likely in order to accommodate for the fact that students had very little prior knowledge of the topics.

It became clear that the course needed a redesign (this was 2015). The starting point was the book "*Language and Computers*" by Dickinson et al. [10]. We started by creating quizzes on the Moodle online platform [12] based on the book as we wanted to stay true to the original intended learning outcomes of the course by providing students a good overview of the field of language technology. Very quickly we realized that we needed to implement practical assignments and included a few regular expression assignments. The final assignment was to look at the lexical associations and frequencies of 'big' and 'great' in a TED-talks corpus. As most students ended up using AntConc for this assignment, and only a third of the students submitted their assignment on time, we decided to implement AntConc and TagAnt assignments on the course for the following year (2016). In 2016 we also introduced the students to some basic command line tools. We faced some technical diffiulties as all students had to get UNIX accounts at CSC[4] to be able to use *grep*.

The very practical assignments with AntConc and TagAnt worked so well that we wanted to give the students more options (in 2017), and included *Python* versions of the *AntConc* and *TagAnt* assignments. Despite the technical diffuculties we had in 2016, we chose to do these too on the same CSC server as the previous year as we felt that the installation of *Python* tools on their own, mostly Windows, computers was going to be more troublesome in terms of technical issues. They still had the option to use *Python* on their own computer though.

This year (2018), the course remained similar to the previous year's version (2017). We focused on making the assignments clearer and providing more practical examples.

[4]Center for Scientific Computing, Finland

We also added lab sessions for each week and instructed students in the use of LaTeXfor the final assignment although this was not a requirement. The steps we have taken have shown themselves to be quite fruitful as measured by student retention rates which have steadily been increasing over the years.

In 2018 a follow-up course was also introduced: *Command line tools for linguists*. This course started the week after the introduction course finished and mainly consisted of students who had just finished the previous course. This course proved very popular and will be repeated in 2019.

3.3.2 New Developments

In 2019 we want to implement short lectures before the computer lab sessions that introduce the topic for that week. We believe this should clarify many assignments and help students focus on the most pertinent information. Some version of these lectures would also be available online.

Another improvement we originally planned on making was using *Jupyter notebooks* for learning code-literacy [25]. *Jupyter notebooks* allow you to share code in small snippets that can be run independently. They can be modified and run in place, and they make the code highly visible. They are also shareable so a teacher can create a notebook and then share it with students. Using *Jupyter Notebooks* should make it easier for students to manipulate code and it requires less setup work by the students eliminating the need to use the services of CSC which should lead to fewer technical problems. However, based on the student feedback (see figure 2, question 2) from 2018, the CSC server was the one aspect of the course that no student was unhappy with, therefore we have to re-evaluate to what extent *Jupyter Notebooks* will be implemented.

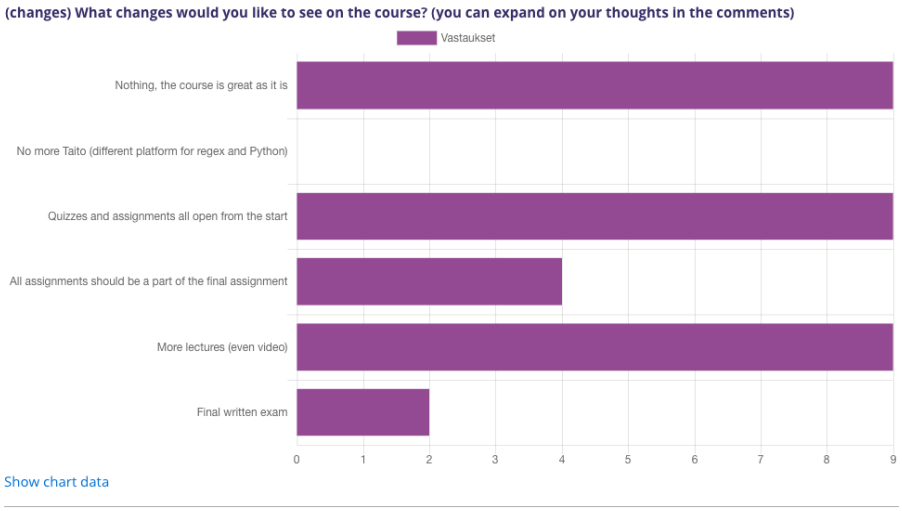

Figure 2: Student-requested revisions to the course.

Additionally, the course will be restructured to make the importance and idea behind the assignments clearer from the beginning. This is done by making all assignments a part of the final assignment. The tasks and assignments of each week will be slightly modified so that they all can be used in the final assignment. The final assignment will also seem like a less daunting task when it is something the students have been working on throughout the course. It should also further reduce the number of students who drop the course in the final stretch.

4 Best-Practice Recommendations

In the previous sections I have discussed the approaches of a specific course and some theoretical implications of teaching digital humanities and computational methods. In this section I will summarize best practice approaches to teaching these methods and the order in which to teach them for students of all levels and varying levels of technical expertise.

4.1 From Online to Blended

Perhaps the biggest change, and challenge, was the change from a fully online course to a blended learning environment. Implementing face-to-face lab sessions was definitely an improvement that the students felt they needed. It also helped us teachers better understand what the students were having issues with; like one of the students complained in their feedback, we did not always understand their questions. On courses such as this one, students often lack the vocabulary and even understanding to describe what they do not understand. This is another reason implementing Jupyter Notebook versions of the *Python* tasks should reduce issues with describing the problem at hand as every student has the exact same code in front of them in the exact same environment. However, if the original course has been an in-class only course, it is vital that a large part of the course material is available online so that the students can take the time they need to go through the material [24].

4.2 Planning a Course in Computational Methods

When planning a course in computational methods or similar to students of traditionally non-computational fields, consider the starting point of the weakest student, now imagine it's worse than that. You will need to explain some very basic concepts to students including what a method and an application is and what the difference between them are. Even when you give them step-by-step instructions with screen-shots and screen-capture videos, there will be a few students who have already fallen behind.

The difficulty lies in setting up the course so that all students will find the topics relevant to their own work, but also in creating an atmosphere of curiosity and exploration. Ideally, students would gain the courage to explore and try independently [13]. There is a difference in attitudes between undergraduate and graduate students as well. At graduate level or beyond, most students seem to have an understanding of the importance of digital methods for their research whereas, based on course feedback, it

seems too many undergraduate students see little benefit in learning digital methods. This discrepancy leads to students acquiring very few computational skills at undergraduate level, only to panic at graduate level when they realize that they could indeed have use for these methods.

A direct challenge in teaching computational methods in this environment is not only reluctant learners, but also technophobia. Even when given step-by-step instructions and screenshots of how to, for example, type in commands in a terminal, some students are so fearful of making a mistake and perhaps even damaging the computer, that they completely freeze. When they normally might use a search engine to find solutions to their problem, what can only be described as technophobia seems to take over, and they simply give up. This is very apparent when looking at the *Help Forum* for these courses.

This leads to a related issue: namely highly varying skill levels. When some students are more familiar with computers, they have an advantage over those who do not. In this day and age, everyone knows how to do some basic tasks on a computer, but they might not have any understanding of even how the file system on their OS (operating system) works, nor what an OS is. Thus, when using a command line interface they often have difficulty ”locating” themselves in a file system that they have to conceptualize in their heads. Understanding file hierarchies are often taken for granted by instructors and many students are embarrassed to confess that they do not understand and then drop the course. Great care needs to be taken to establish a baseline of knowledge before jumping into more difficult tasks. Making the students actually learn something without making it too difficult becomes the next challenge.

Especially with undergraduate courses, one needs to be very careful with how course topics are presented. Some students, often those with lower levels of computer-skills, get anxious the moment programming is mentioned or when they see a terminal window. Students have timidly asked me after the first lecture if they'll be required to learn programming because if they have to, they will drop the course. To this I reply something along the lines of ”not to worry, all you have to do is some copy-pasting and editing existing code using very clear instructions”. The student who asked not only stayed on the course, but for the final assignment chose to write a computer program in *Python*.

4.2.1 Course Content

The first and most important thing the students need to learn is to consider texts in a new light. A text on a computer is not just the meaning content. There are many different encodings, file formats, scripts and such that all need to be handled differently when using computational methods. Ideally the students would learn to instantly identify and differentiate between e.g. tsv, csv, and txt files at a single glance. They should also be able to fix a wrongly encoded file. Another important aspect is for students to understand that word boundaries do not equal white-spaces or line-breaks. This is often successfully demonstrated by teaching regular expressions with appropriate tasks.

Again, regular expressions look like code and can therefore be scary to some students at first glance. This is why I always start teaching regular expressions by telling the students that they likely already use regular expressions every day, for example in

library searches or Google searches. There are many good interactive regular expression tutorials online, and I ask that the students do one of them so that they learn of the existence of the different basic sequences, quantifiers, and metacharacters. After that I give them a few real-life problems to solve that force them to consider the difference between white-space, line-breaks, and word-boundaries. This can be as simple as asking them to find all numbers denoting years in a text. Once the students have a basic understanding of how regular expressions work, they learn to apply those skills using *grep* and *sed*. Often times it is not so important that students master something like *sed*; what is important is that they know such a tool exists and where to find more information on how to use it once they do need it.

Once the students are working with *grep* or *sed*, it is a short jump to learn how to navigate the computer in a Unix command line environment. Even if the students were to never use a command line interface again, however unlikely, they will learn how a computer file system is structured without any visual cues from a GUI, and without using a mouse. They will learn to visualize the structure of a computer in their heads [29].

The next step is to show some actual code to the students and explain what it does. This does not necessarily even have to include the basics such as loops or data types. Instead, something simple like reading in a text, doing some small change or analysis with the data and then saving that new data in a new file is a great teaching example. It allows students to see the file path and understand where, how, and why it needs to be modified if they want to run the same code on their own computer.

After the students have a basic understanding of what code looks like and how it works, they are given some scripts that they have to run and modify. This often leads to error messages of various types and possibly the most important lesson: how to independently find help online. It is important that students realize they can use Google to find solutions to an error message they have received. Very quickly they will find themselves on StackOverflow. Once the students reach this point, many start wondering about where they can learn to code properly. I refer them to the appropriate follow-up courses, but also encourage them to learn *Python* on *Codecademy* [32] and other online course platforms.

The final step is learning to use *Python* for one's own project. Therefore it is imperative that the assignments on the course are all meaningful and designed to help the student understand the content rather than to evaluate how much the student has learned. An evaluation can be completed with a final project report where the students are either given a dataset, or asked to design their own project. For an undergraduate course, it is often best to start with a pre-determined dataset as many undergraduates do not yet quite understand how to limit the scope of their projects.

4.3 Context

It is highly recommended to create polls or some other form of feedback several times during the course to give the students an opportunity to anonymously evaluate their learning and subsequently the teaching. These polls will be very helpful when introducing the next problem set or topic. The course should have minimal traditional quizzes and exams [36]; depending on the intended learning outcomes of the course,

some might be necessary, however, in general all assignments should be designed to help the student learn rather than test how much they have learned [22, 28].

As for planning an actual curriculum, it is important that there is a well-thought out progression of courses. That is, the follow-up course to a basic course in computational methods (a course similar to Introduction to Language Technology), needs to go deeper into the subject matter introducing new concepts and broadening the understanding of previous topics. However, the step up should not be so steep that a student who has only done the previous course cannot genuinely participate. This is of course the very essence of university pedagogy regardless of discipline (see [38].

These courses are rarely followed up with appropriate indermediate-level courses. Either the next course is of the same beginner-level, frustrating students because they are essentially doing the same course over again without gaining any new skills or knowledge. Alternatively, the follow up courses jump up to advanced-level or pure CS-courses, again frustrating students, but now because the students who only took the beginner course have no realistic way of following the course topics or completing the assignments. Doubly so since their special interests are not taken into consideration.

5 Conclusions

There is much to consider when designing a course that teaches computational methods to humanities students. Hopefully this paper will give some practical ideas to those who do design these courses. In a society that is becoming more and more digitalized, it is increasingly important for students of all disciplines to understand the underlying concepts that steer the digital society around them.

Undeniably, some of these experiences are limited to Finland and even to the University of Helsinki, however, I believe that most of these recommendations are valid for other countries and universities as well. After all, the struggle to effectively teach computational methods to humanities students seems to be a universal one [14, 16, 21, 26, 29, 35].

References

[1] External grants & funding: EADH - The European Association for Digital Humanities. https://eadh.org/support/external-grants-funding.

[2] Funding & Opportunities. http://digitalhumanitiesnow.org/category/funding/.

[3] Funding: EADH - The European Association for Digital Humanities. https://eadh.org/news/category/funding.

[4] Project Gutenberg. http://www.gutenberg.org/.

[5] ABRAHAMS, D. A. Technology adoption in higher education: A framework for identifying and prioritising issues and barriers to adoption of instructional technology. *Journal of Applied Research in Higher Education 2*, 2 (2010), 34–49.

[6] ANTHONY, L. AntConc: A learner and classroom friendly, multi-platform corpus analysis toolkit. *Proceedings of IWLeL* (2004), 7–13.

[7] BARTLETT, A., LEWIS, J., REYES-GALINDO, L., AND STEPHENS, N. The locus of legitimate interpretation in big data sciences: Lessons for computational social science from-omic biology and high-energy physics. *Big Data & Society 5*, 1 (2018), 2053951718768831.

[8] CHAIKLIN, S. The zone of proximal development in Vygotsky's analysis of learning and instruction. *Vygotsky's educational theory in cultural context 1* (2003), 39–64.

[9] CROXALL, B., AND WARNICK, Q. Failure. *Digital Pedagogy in the Humanities: Concepts, Models, and Experiments. Modern Languages Association* (2016).

[10] DICKINSON, M., BREW, C., AND MEURERS, D. *Language and computers*. John Wiley & Sons, 2012.

[11] DOMBROWSKI, Q., GNIADY, T., MEREDITH-LOBAY, M., THARSEN, J., AND ZICKEL, L. Research computing's demand for humanists, and vice versa. In *Digital Humanities 2017: Conference Abstracts* (2017). https://dh2017.adho.org/abstracts/DH2017-abstracts.pdf.

[12] DOUGIAMAS, M., AND TAYLOR, P. Moodle: Using learning communities to create an open source course management system. In *EdMedia: World Conference on Educational Media and Technology* (2003), Association for the Advancement of Computing in Education (AACE), pp. 171–178.

[13] DUFVA, T. Creative coding at the arts and crafts school Robotti. In *Digital Humanities in the Nordic Countries 2018* (2018), CEUR Workshop Proceedings.

[14] EARHART, A., AND TAYLOR, T. Pedagogies of race: Digital humanities in the age of Ferguson. *Debates in the digital humanities* (2016).

[15] FORTE, A., AND GUZDIAL, M. Motivation and nonmajors in computer science: Identifying discrete audiences for introductory courses. *IEEE Transactions on Education 48*, 2 (2005), 248–253.

[16] GNIADY, T., AND WERNERT, E. An open reproducible method for teaching text analysis with R. In *Digital Humanities 2017: Conference Abstracts* (2017). https://dh2017.adho.org/abstracts/DH2017-abstracts.pdf.

[17] GRAHAM, C., CAGILTAY, K., LIM, B.-R., CRANER, J., AND DUFFY, T. M. Seven principles of effective teaching: A practical lens for evaluating online courses. *The Technology Source 30*, 5 (2001), 50.

[18] GRANDELL, L., PELTOMÄKI, M., BACK, R.-J., AND SALAKOSKI, T. Why complicate things?: Introducing programming in high school using Python. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52* (2006), Australian Computer Society, Inc., pp. 71–80.

[19] HARLEN, W., AND JAMES, M. Assessment and learning: Differences and relationships between formative and summative assessment. *Assessment in Education: Principles, Policy & Practice 4*, 3 (1997), 365–379.

[20] HEW, K. F., AND CHEUNG, W. S. Students' and instructors' use of massive open online courses (MOOCs): Motivations and challenges. *Educational research review 12* (2014), 45–58.

[21] IDE, N. M. Computers and the humanities courses: Philosophical bases and approach. *Computers and the Humanities 21*, 4 (1987), 209–215.

[22] JENSEN, J. L., MCDANIEL, M. A., WOODARD, S. M., AND KUMMER, T. A. Teaching to the test... or testing to teach: Exams requiring higher order thinking skills encourage greater conceptual understanding. *Educational Psychology Review 26*, 2 (2014), 307–329.

[23] KARLIN, J., SMYTH, P., ZWEIBEL, S., AND GOLD, M. K. DH Box:. In *Digital Humanities 2017: Conference Abstracts* (2017). https://dh2017.adho.org/abstracts/DH2017-abstracts.pdf.

[24] KETTULA-KONTTAS, K., AND MYYRY, L. Understanding forest sector ethics and corporate sustainability through blended learning. *Blended Learning in Finland* (2010), 65.

[25] KLUYVER, T., RAGAN-KELLEY, B., PÉREZ, F., GRANGER, B., BUSSONNIER, M., FREDERIC, J., KELLEY, K., HAMRICK, J., GROUT, J., CORLAY, S., IVANOV, P., AVILA, D., ABDALLA, S., AND WILLING, C. Jupyter notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (2016), F. Loizides and B. Schmidt, Eds., IOS Press, pp. 87 – 90.

[26] KOCH, C. On the benefits of interrelating computer science and the humanities: The case of metaphor. *Computers and the Humanities 25*, 5 (1991), 289–295.

[27] KOKENSPARGER, B., AND PEYOU, W. Programming for the humanities: A whirlwind tour of assignments. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2018), SIGCSE '18, ACM, pp. 1050–1050.

[28] LARSEN, D. P., BUTLER, A. C., AND ROEDIGER III, H. L. Test-enhanced learning in medical education. *Medical education 42*, 10 (2008), 959–966.

[29] RAMSAY, S. Programming with humanists: Reflections on raising an army of hacker-scholars in the digital humanities. *Digital Humanities Pedagogy: Practices, Principles and Politics* (2012), 217–41.

[30] RASKIN, J. Flow: a teaching language for computer programming in the humanities. *Computers and the Humanities 8*, 4 (1974), 231–237.

[31] RUSHKOFF, D. *Program or be programmed: Ten commands for a digital age*. Or Books, 2010.

[32] SIMS, Z., AND BUBINSKI, C. Codecademy. *http://www.codecademy.com* (2011).

[33] SINCLAIR, S., ROCKWELL, G., ET AL. Voyant tools. *URL: http://voyant-tools. org/[September 5, 2016]* (2016).

[34] TARAS, M. Assessment–summative and formative–some theoretical reflections. *British journal of educational studies 53*, 4 (2005), 466–478.

[35] TRACY, D. G., AND HOIEM, E. M. Access to DH pedagogy as the norm: Introducing students to DH methods across the curriculum and at a distance. In *Digital Humanities 2017: Conference Abstracts* (2017). https://dh2017.adho.org/abstracts/DH2017-abstracts.pdf.

[36] UDOVIC, D., MORRIS, D., DICKMAN, A., POSTLETHWAIT, J., AND WETHERWAX, P. Workshop biology: Demonstrating the effectiveness of active learning in an introductory biology course. *AIBS Bulletin 52*, 3 (2002), 272–281.

[37] VAN ES, K., WIERINGA, M., AND SCHÄFER, M. T. Tool criticism: From digital methods to digital methodology. In *Proceedings of the 2nd International Conference on Web Studies* (2018), ACM, pp. 24–27.

[38] VYGOTSKY, L. Zone of proximal development. *Mind in society: The development of higher psychological processes 5291* (1987), 157.

[39] WILIAM, D., AND BLACK, P. Meanings and consequences: A basis for distinguishing formative and summative functions of assessment? *British Educational Research Journal 22*, 5 (1996), 537–548.