Python for the Past: How to teach Python to modern archaeologists

Poelenije, Cas

Python for the Past

How to teach Python to modern archaeologists

C. A. Poelenije

Python for the Past

How to teach Python to modern archaeologists

C. A. Poelenije, S1686097

Graduation Project Applied Archaeology, 1084VTMAY

Dr. Tuna Kalayci

Leiden University, Faculty of Archaeology

Zevenbergen, 15-06-2024, Final version,

Acknowledgement

To all whom supported me when I needed it most,

I thank you.

You know who you are.

Table of contents

Chapter 1 Introduction

1.1 Introduction

Archaeology studies the human past through unearthing, examining, and analysing material culture. A large part of archaeology revolves around the influences that knowledge and technological innovations have on human sub- an existence. Archaeology as a discipline also evolves along with technological changes and advancements. In our current age we have turned to using computer-based tools to help enhance the way we process and acquire data. Some examples of this are digitizing databases and datasets or using Geographic Information Systems (GIS) for spatial analysis, predictive modelling, etc. These digital methods have created so many more possibilities for archaeological research or methods within it. Even to a point where it has become an essential part of archaeology. With it being non-destructive and repeatable we can adjust our methods and retry, something that for example with an archaeological excavation would be impossible.

In order to create or interact with such computer-based tools, the skill of coding can be very useful. There are, however, many different programming and scripting languages that can be used in the context of the archaeological field. These languages all have their own strengths and weaknesses when it comes to certain topics and desires. R is a coding language that excels in statistical analysis and data processing for example, where SQL is a language designed to work with large relational databases. C++ is a language that is known for creating high performance applications with efficient processing that can be helpful for making standalone applications or programs. There are still other examples of how different coding languages can be used and useful in the archaeological field. However, for this research, the focus has been put on Python.

Python is an advanced scripting language that is used for, or can interact with, many different programs. It is relatively simple to learn and open source. There are many third-party additions for specific goals and purposes available as well. Python can be used to for handling large or complex datasets, visualising results, automating digital tasks and much more. Python has a lot of overlap in its potential functionalities with other programming languages. This makes Python a great language to start one's computational toolbox with, since it allows for a wide variety of applications.

Because of the potential in the synergy between Python and the archaeological field, the main research question for this thesis is: How to best teach Python in an archaeological context? In order to fully answer this question, multiple sub questions are posed as well. These are:

- Why specifically Python?
- What can Python add to the archaeological field?
- What are common issues associated with teaching Python?

- How useful can GPT models be when learning Python?

In addition to answering these questions, part of this research is also setting up a course plan for teaching the basics of Python, supported by the findings of the study. This makes the goal of this thesis to develop a pedagogical framework for, and show archaeological relevance of the basic skills of Python coding.

1.2 Examples of Python in archaeology

Python is already being used for a variety of projects and research within archaeology. It can be used to create standalone tools, but also to enhance existing ones. For example, Francisci (2021) uses a Python script to classify archaeological data based on geometric intervals for visualisation in QGIS. Which means that archaeological data in question could be presented in QGIS better than it could with the default options QGIS offers. In doing so it improves the functionality of another often-used computer-based tool, that being QGIS. Python can also be used to make data more accessible or easier to work with. An example of this is given by Bird et al. (2022). They used Python in their efforts to make entries from different large datasets of radiocarbon data more homogenous. This eliminated some of the issues with the interoperability of the data. Python's role in this was to help clean the data automatically. This entailed the application of certain steps to all data in the dataset. Something that would be tedious and incredibly time consuming for a person to do considering the large size of the dataset.

There are many third-party additions available for Python that make it easier to execute certain actions. These are very useful since they add a lot of functionality for specific goals, without it taking up a lot of extra time from the person looking to use it. Considering this it is no surprise that there are also packages made to help with archaeological analysis. One such package is PyLithics (Gellis et al., 2022) which is a Python package that captures data from drawings of different flint flakes.

These three examples showcase the exciting possibilities for the application of Python in archaeology to facilitate research in very versatile and specialised ways.

1.3 Available education

We have established that, even though it might not be the first thing one thinks of when they hear archaeology, Python has great potential in the archaeological field. In order to not only create programs or scripts with Python, but also to understand those of others, it is important that archaeologists learn Python. It is however rare to see this, or a similar, course in curricula of archaeological studies. In the study guides of different archaeological bachelor and master studies in the Netherlands and Belgium I looked for classes teaching Python or related subjects. University Leiden offers a specific class where Python is taught in combination with a small degree of statistics.

Saxion in Deventer and VU in Amsterdam both have a course in their bachelor that is digital archaeology, however this covers basic databases, GIS, and statistical topics. No dedicated coding is present besides limited SQL. The VU in Amsterdam does offer a Digital Archaeology master track, however I could not find any mention of (Python) coding in their study programme. In Belgium the image is not that different. Gent offers no more than a statistical analysis course. The university of Leuven also does not offer any courses related to coding.

1.4 Reading guide

This thesis is structured in chapters and sub chapters where relevant. The first chapter after this introduction will be the chapter of methods and framework. This chapter will start by explains some important and relevant terms. Next it will discuss why exactly Python is the chosen scripting language for this thesis, followed by a more in depth look at examples of Python being used in the archaeological field. After this there will be sections explaining the research methodology and how the course plan will be formed. Finally this chapter will end with a short description of the profile for whom this thesis is made and designed.

The third chapter goes into the literature. First the general term of pedagogy will be explained before this term will be narrowed down to the actual context of computational thinking and writing code. This is then followed up by presenting relevant literature for different teaching methods and techniques for similar topics.

The fourth chapter will discuss what the actual course plan consists of. This will include what it is, what the covered topics will be, how it is structured, and finally what type of supporting materials are made.

After this chapter there will be the discussion of the thesis. This chapter will start off by presenting and discussing the findings of the research. This is then followed by a discussion of the course plan as well as a discussion of the use of GPT models in learning Python. The last section of this chapter will go into some of the challenges that were encountered in writing this thesis.

Chapter six is the final chapter. The first part of this conclusion will answer all sub questions before using those answers to answer the main research question. Next will be a section on the significance of the research and some practical implications. And the final part will be about limitations of the research and some future recommendations.

Chapter 2 Methods and Framework

The framework for this thesis will start by addressing and explaining some important and reoccurring terms. After this there will be a section that explains why specifically Python is chosen as the focus language for this research in relation to archaeology as well. To bring this more into context there will be three cases which will highlight different practical applications of Python in the archaeological field. This is then followed by a section explaining what this research will produce, together with how it will do so, and what the intended profile of potential readers and users might be.

2.1 Important terms

In order to be able to fully explore the merits of Python scripting in archaeology, we first need to define some of the important terms that will be used in this thesis.

The first term is **digital archaeology**. Digital archaeology is the part of archaeology that deals with digital methods and tools for collecting and analysing of archaeological data. These methods and tools include, but are not limited to, using GIS programs for spatial analysis, modelling, statistical analysis.

Secondly, what is **programming** and what is **Python** in relation to that? According to the Cambridge Dictionary, programming is the activity or job of writing computer programs. Programming is done by setting instructions for a system to follow. This means that a file with different lines of code in a specific coding language will execute or perform certain tasks on a computer or system. Python is a free and open-source advanced scripting language. The difference between scripting and programming is that scripting refers to making code that interacts with programs or automates tasks, where programming revolves around the creation of programs. Python is known for being flexible and having many useable third-party packages, Python code that is already written by someone else, to draw from, while still having a very readable and clear syntax. It also works on various platforms and brands of operating systems, making it suitable for transferring programs or code between different computers.

Open source means, among other things, something that is publicly accessible and modifiable. When it comes to software it is measure to some points. These are (https://opensource.org/osd, data gathered 11-06-2024):

- The software may be distributed freely.
- Programs or software must include accessible source code, allowing people to modify it where needed.
- Modified works should be allowed to be distributed.
- Integrity of author's source code must be maintained.

- The product cannot discriminate against persons or groups.
- The product should not discriminate against fields or endeavours.
- The rights attached to the licence must apply to all to whom it is redistributed.
- License should be independent of specific products.
- License should not restrict other software.
- License must be technology neutral.

It is important to work within the ideas of open source principles because it can help other people, it stimulates quicker innovation and it allows for the use in education and development without depending on certain licenses or program packages.

Archaeology already employs digital methods and tools for a variety of different uses within research. However, it can certainly benefit from being able to use Python programming more as well. By writing a program for a certain topic or project it is possible to make repeatable data analyses that are specifically fit for the dataset at hand. Besides that, Python also has access to a series of libraries that can help with spatial analysis. Another example of the use of Python in archaeology is that it allows archaeologists to read the code that other programs are built on. In doing so, better understanding how a program works or even how to change it or write a plug-in or addition to said program.

A **GPT model** is a Generative Pre-trained Transformer (GPT) model. This is a type of 'Artificial Intelligence' that is trained on a large dataset with different subjects and written in different languages. This results in a model that possesses a lot of information on a wide variety of topics. Models like this can generate responses to inputs based on the knowledge and languages it knows. Its great understanding of language means that it can be asked questions in a very informal way and still create answers that explain complex subjects or processes. Because of this it is easy to ask questions in a way that does not require us to possess the knowledge of jargon or terms for the model to understand our requests. The chat-like way the model is available also allows for multi-step dialogue. This means that if we receive an answer, the model still has access to all its earlier answers and all our earlier questions and provided information. Although there is a limit to what it can 'recall', that limit is quite large and improves with the technology advancing. The importance of this feature is that we can ask for further elaboration on given answers, ask for simplification of used terms, or even to provide the answers in a different format or language.

GPT models also have some downsides or issues that users should be aware of. One of these issues is the possibility of bias in generated responses. As these models are 'taught' subjects based on the information it is fed, there is a risk that the bias on the subject from some of the training material will be present in the responses to prompts. An issue that can also be frustrating, is that models like

these will not always indicate when something is unknown to it. On multiple occasions in my own experimentation, the model, in this case ChatGPT, made up data and answers that were inaccurate or, even worse, fully untrue. If this was pointed out in these occasions, the model would 'apologise' and try to generate another response. This can, and has, also lead to wrong reference and fact-checking if the model is asked to do so but finds no real references. This also leads into another issue, namely the issue that plagiarism can occur unknowingly. If there is a reliance on a GPT model for generating text for any textual work, there is no real way for the model to accurately, consistently, and reliably be able to provide the proper sources and references. Having to find these can be hard and time consuming, thereby severely diminishing whatever resources one thought to have won by using such a model.

The issues named here are however not meant as reason to discard the use of GPT models. It serves to indicate that the use of these models is not without troubles and thinks that need to be considered.

2.2 Why Python?

Python is considered a scripting language that is both relatively easy to learn and one of the most versatile scripting languages available. This makes it a very suitable language to learn for any field of research. The availability of a large number of third-party packages and modules means that the potential of using Python in different scenarios and projects is nearly limitless.

Every digital language has a syntax. Think of this a similar to the grammar of a spoken language. Each one has different rules, and some are easier to understand than others. In things like scripting languages the syntax refers to, among other things, what symbols should be used and when, how many spaces or tabs should be before or between elements, and if one needs to indicate the end of a line of code. The syntax of Python is relatively simple and clear to understand. The 'structure' of many elements in Python looks quite similar, making for easy recognition and reading. This also means that it is easy to find examples from different sources to help or inspire potential codes and scripts.

2.3 Python's relevance to archaeology

Python is quite easy to use for data analysis and processing. Packages like NumPy and Pandas make this very accessible and easily automated as well. This will already help with dealing with multiple datasets that are presented or gathered in the same format. This allows Python to be used for many tasks related to data analysis. There are, however, also prewritten packages of Python code that are written for archaeological purposes. A search on the website pypi (https://pypi.org/), a website that shows easily available projects with Python code that other users can access, returns already 36 projects for the term 'Archaeology'.

In order to create a better understanding of how Python can be applied in the archaeological field, three case studies will be discussed. Important to know is that the given examples here exceed the level of Python programming that is aimed for in the course and scope of the research. The achieved level should however allow people to be able to read and understand the code that is written by the creators of the code in the case studies. In this way they would be able to adjust or generalise certain things that are being done. And in that still allowing people to understand the steps that were taken. This is also the reason that the description and exploration of these case studies is mostly about the goals, results, and theory, and will not dive into the actual code and scripts. The case studies serve to present different possibilities in different aspects of the archaeological field, not as direct examples of how to write code.

Of the three cases that help illustrate some of the uses of Python programming in archaeology, two are modules or packages specific to archaeological application and context. The third however is more 'normal' Python as it discusses how it is used for large scale data wrangling. This is also to show that, although more in depth scripts and packages can be written for archaeology, even a more generalised application of Python scripting van save the practitioner time by automating large scale tasks.

For our first example we will look at PyLithics. "PyLithics is an open-source, free for use software package for processing lithic artefact illustrations scanned from the literature." (Gellis et al., 2022, p. 1). Archaeological literature on lithics (stone tools) if often accompanied by detailed drawings of the artefacts. The program is helped in being able to work with these drawings because they have established rules and conventions regarding to how artefacts are drawn, scaled, rotated, etc. (Gellis et al., 2022, p. 2). PyLithics uses multiple computer vision techniques, a more advanced form of machine learning that deals with visual recognition, to gather measurements and other data from the drawings of the artefacts and turns it into useable data for researchers (Gellis et al., 2022, p. 1). In principle, what the software does is not something that could not also be done by researchers themselves. However, it would be significantly more time and labour intensive. Especially compared to the speed with which the software will be able to process a multitude of images. Although creating and 'training' such a model is quite difficult and time intensive, using the tool is not that difficult. The available online documentation (Gellis et al., 2022) also provides guidance and instructions on how to use the tool. The documentation also shows examples of data output and conventions to get the best output out of PyLithics.

The second case study revolves around a lack of a proper classification possibilities in QGIS (Francisci, 2021), a Geographic Information System that if often used by archaeologists for modelling and visualisation. Due to the distribution of the data that was being worked with, right skewed, none

of the standard data classification methods was applicable. Due to the open source nature of QGIS and the Python implementation, Francisci wanted to solve this with said programming language. Francisci successfully included the new and applicable way of classification to the QGIS model using a Python script (Francisci, 2021, p. 6). Interestingly this case also supports the mentality of this research, that code-literacy is already a powerful skill in its own right. Francisci describes broadly how he went about writing his script and, in doing so, mentions different people whose code he used and tweaked in order for him to arrive at a script that had the desired result and functionality (Francisci, 2021, p. 5). Francisci refers to this process and its value with an analogy in his conclusion: "Open-source code, and a wide community of users that share their works and suggestions, allow us to create the tools we need on our own, similar to an ancient craftsman building his own tools. In this way, we can understand how the software functions work, and we can use them correctly, moving from an 'unaware click' to an 'aware' click." (Francisci, 2021, p. 6). This sentiment is also upon which this research and the accompanying course are written for the archaeological context.

The third and final case study shows how to deal with differences in datasets over many samples and how Python programming has helped in dealing with this. The case study of p3k14c (Bird et al., 2022) deals with a global scale database of carbon 14 dates. Many different labs have conducted c14 dates for different contexts and with different conventions. This creates the issue that all these different datasets cannot be combined into one large database. Their formatting would make that unreliable and unreadable. In order to make sure that these datasets could be combined, many changes and much data 'cleaning' needed to be done. A part of this was done with the help of Python programming. This 'scrubbing' process as it is called, removes faulty entries based on certain criteria with Python. The removed entries are then saved elsewhere, along with a reason for deletion. The Python script would go through this process in five broad steps. The first step is to remove records from laboratories that are unknown, followed by a standardization of coordinate data relating to location. The third step deals with duplicate data. After that there are a set of 'miscellaneous' cleanings that are done by the script. And finally the script would obfuscate any data where needed to protect certain sites (Bird et al., 2022, p. 6). None of these applied steps are necessarily something purely archaeological. They can be considered actions or tasks that would be done in any large scale data oriented project. The size of this database is, according to the publication, 180,070 archaeological carbon ages (Bird et al., 2022, p. 2). But in automating these tasks with programming, there is a lot of time that is saved, which otherwise would take up many days of many researchers.

These three case studies illustrate the various benefits of including Python in archaeological research, ranging from the ability to develop new methods to gain data from publications to automatization of otherwise time-consuming tasks.

Ingraining the use of Python more in archaeological research can also help the field keep up with future and current digital developments. It might also help lowering the threshold for getting into more digital technologies and methods.

I find it important that the human element should not be fully removed from the research, when talking about automation. For this might lead to a lack of understanding of the processes that happen. This is, however, less a problem when the automation is applied by someone who understands how the process works on a detail level. Python should be seen and used as a tool. It can make research be more efficient, but there are risks when the researcher is not involved in this process.

2.4 Method

The main method for this research is using existing literature to best support methods and solutions for teaching and learning Python. Based on literature we can establish what some of the most common issues are with teaching and learning Python, and find ways to either circumvent or solve these. This will be done by comparing multiple teaching philosophies, especially in the context of digital skills such as coding and working with computer logic. Besides that we will also establish what skills are important and which skills seems to be most difficult within this context.

Although the main focus of this thesis is on the Python language, the literature study will also include a wider scope. This refers to the inclusion of pedagogy for computational thinking. Computational thinking being a major part of any digital skill.

The finding from this literature study can then be incorporated in the course plan and structure. In this way creating a course plan that can serve, if not in its entirety as a course plan, at least as a template. The findings are also important as they might serve as reference material for educators in this, or a related, topic for dealing with issues and troubles that may occur.

2.5 The course plan

The course plan that accompanies this research will be discussed fully later in the paper. Here we present a short summary of topics that form the basics of Python. Also, the broad set-up or structure of the course will be discussed.

The course will start from the very earliest parts of working with Python. This also means covering IDE's (Integrated Development Environment) and their differences. And IDE is a program that helps the user in writing code by providing different features such as syntax check or easy documentation.

However, this differs between IDE's. Then focus will be put on learning to understand the syntax of Python and the different datatypes and operators. Also how do the operators interact with different data types. This will also include lists and dictionaries. From there loop control will be an important subject, followed by understanding what a function is and how to define/make one. After understanding functions an introduction will be made with third party packages and their wide implementation and potential. To help create good interactions with their scripts we will also go into dealing with errors. An alternative way of programming, object oriented programming, will also be discussed. Finally there will be a section on implementation and ethical considerations of the use of A.I. in a coding context.

These topics will be taught by means of presentations and individual small assignments. This way any given theories can be tested and practiced by means of suitable assignments. The practical part is important as it helps people realise how and what they are doing.

Besides the theory and practice parts discussed, small amounts of more complex code will also be shown executed step by step in order to teach people how the code gets followed and executed.

The core knowledge of Python will be very important in order to learn more specialised applications of Python in a later stage.

The course structure will be as such that parts of the course can be taken separately to develop the related skills. This can be useful for those who already have a basic understanding of Python. This does mean that for each sub-subject the basics will not be repeated, at least not in depth. In doing this it allows for teaching of the more specialised matter quicker. If there is already a knowledge of Python, there will be no wasted time on too much-repeated subject matter.

2.6 Profile

This thesis and course plan are for archaeologists who want to get to grasps with the basics of Python, or those that want to use it as and aid in teaching it to others. The main goal is to help introduce and teach Python in the field of archaeology. But, with the widespread potential of Python, this material does not have to be limited to that field of study.

It can help people understand and read Python with help of examples that are familiar from actual experience with archaeology, find strategies for overcoming common troubles with learning and teaching coding, or serve as a refresher for those who need it.

I hope this course can reach and help researchers who are looking for a smarter or quicker way for things to be done, without losing reliability. Or those that have ideas on how to build upon existing tools for research and development within archaeology, or any field for that matter.

Chapter 3 Pedagogy

This chapter goes in depth on the pedagogy of teaching Python and related skills. In order to do this it will first introduce and explain what pedagogy is. Then the literature in this chapter discusses a modern rendition of Bloom's Taxonomy of Learning to help establish which skills are most difficult to learn and how to indicate different skill levels. There will also be a discussion of gamification and active learning methods with regards to learning and teaching Python.

3.1 What is pedagogy

Pedagogy refers to "the art, science, or profession of teaching" (Merriam-Webster, n.d.). It is a term that encompasses methods and techniques of teaching people. There are many ways and approaches to facilitate the best and most practical education. Pedagogy is not only important for which topic is being taught, but also how it is being taught. Some things to take note of when planning to teach something are, for example, the environment in which the subject will be taught, the planning and length of individual lessons and choosing which subject to teach. Although the most optimal context can be different per student, there are certain things that can be objectively beneficial to nearly all students.

It is because of these different preferences for different people, that certain methods have been reworked or created because they get the most optimal results for the most people. These methods can however differ based on the topics and the related issues to those topics. For teaching Python, or programming in general, there are some often reoccurring issues and difficulties. These will be discussed and addressed later in this chapter. But these challenges do require, to a certain degree, that some of the more conventional methods and rubrics need to be slightly adjusted.

For each specific topic that needs to be taught, there are specific issues, troubles, and challenges. Therefor it is important to consider the pedagogy and teaching strategies of teaching (Python) programming and, by extent, computational thinking. The next section will discuss some methods and ideologies that can help teach programming and deal with some common issues. Specifically, the topics discussed next will be a programming-oriented variant of Bloom's Taxonomy of Learning (Selby, 2015), gamification elements that can help keep students engaged and motivated, and some active learning strategies when it comes to Python programming.

3.2 Programming and Bloom's taxonomy

In 1956 Benjamin Bloom, along with others, published the Taxonomy of Educational Objectives (Bloom et al., 1956). This taxonomy was a framework for educational goals that is often referred to as 'Bloom's Taxonomy'. The framework created categories and subcategories that would be put in a line to indicate both the current level of a person, as well as what skills and type of knowledge would

be the next step. Since its conception, and up to this point in time, Bloom's Taxonomy has been revised and changed as time progressed, but also as it was applied to different contexts and subjects. The most well-known revision is from 2001 (Anderson & Krathwohl, 2001) and makes the model more approachable by including verbs to indicate how a category of the taxonomy would manifest. This chapter will use a more recent study that creates a variant of Bloom's Taxonomy for computational thinking and programming. The reason this model is introduced in this research is because it can be used to estimate a student's level of knowledge, as well as identify an order in which subjects and topics can be taught.

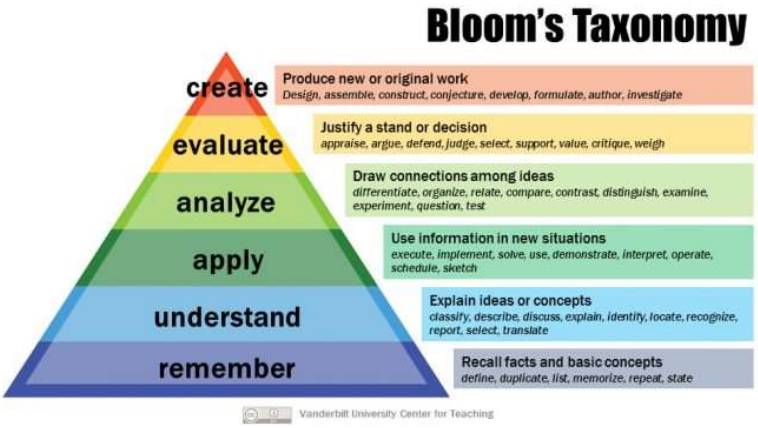

Figure 1 A modern visual representation of Bloom's Taxonomy of learning (Armstrong, P. ,2010)

Selby (2015) explores the degree as to which the taxonomy of Bloom can be applied to computational thinking and programming based on the views of 255 participants, of which the majority identified themselves as educators. In doing so they first establish what skills are a part of computational thinking, followed by establishing which skills, both in terms of programming and computational thinking, are considered to be more difficult for beginners to master.

3.2.1 Computational thinking

Since both the concept of computational thinking and pedagogy are broad terms, Selby (2015) defined specific skills as included in this specific definition of the term. These skills are abstraction, algorithm design, decomposition, evaluation, and generalisation.

Abstraction, as defined by Wing (2008), is the process of determining which details can be ignored and which need to be highlighted. This definition however feels rather crude and could maybe better be interpreted as; the skill to acknowledge which elements have a higher or lower importance. It is a very important skill for problem solving and is to such a degree important that it "…underlies computational thinking" (Wing, 2008, p. 3718).

With the term algorithm design, we refer to the skill of being able to break a process down in smaller sequenced steps. This way systems or programs are able to execute tasks reliably and consistently.

Decomposition, the breaking down of a problem into smaller parts, is important for computational thinking to make larger and more complex issues easier to solve and to deal with.

Evaluation in computational thinking defines the act of evaluating the resources needed and efficiency achieved by a certain product. In context this will refer to things like runtime and storage space.

The final skill mentioned above is generalisation. By generalisation in this context we mean the ability to provide solutions to problems in more generic terms, so it can be applied to similar problems with different contexts (Selby, 2015, p. 81). In modern context a good practical example of this is looking on the internet how people dealt with similar problems or errors. Since these will very rarely match personal context, it is important to be able to 'translate' these solutions.

These terms are here named and described in alphabetical order, however based on their descriptions one should be able to see how these terms and concepts work together and lead into each other in different ways in an actual workflow of solving an issue.

3.2.2 Programming

Selby (2015) starts by identifying some of the larger potential issues that make learning programming more difficult for beginners. The first of these issues is an inaccurate understanding of how computers execute programs or commands (Selby, 2015, p. 82). Ma, et al., researched if the use of a visualisation could help with this problem. They found that it helped create understanding for concepts, and especially even more complex concepts. And it also seemed to create excitement and stimulate engagement (Ma, et al., 2011, p. 77-78). This is also seen in recent programming 'languages' for children. Programs such as Scratch (Jr.) allows children to make things like games or animations with a visual drag and drop system. Allowing them to work with understanding the sequence of processes and actions, without having to fully understand the code or syntax, which are 'hidden' from the children anyway.

The second issue they discuss is the inability to properly read, trace, and write code. Reading and writing code speak mostly for themselves in what they entail, but tracing code means being able to follow what the code does and where in the code that action is written. This is also very helpful in terms of figuring out where issues or errors might occur. Lister et al. (2009, p. 164-165) came to the conclusion that in order for someone to be able to write good code, they also need to be able to trace and explain code. They also found that effective programmers could explain code without necessarily being able to describe line by line what the code was doing. The ability to read and trace code also is a very valuable tool when considered in the context of Open Science principles for coding. The principles of Open Science have as goal to help make research more collaborative and strives to have data, software and more be accessible and reusable as early as possible (Nederlandse organisatie voor Wetenschappelijk Onderzoek, n.d.). A part of this is means that there should be a large amount of code available to read and use as examples for improvement. Coming in contact

with these principles can also help establish value in the Open Science way of thinking early on, since this way of working is designed to help the community grow better and quicker in terms of code and research by making elements early and easily accessible.

The third and final large issue Selby (2015) mentions in programming is an inability to understand high-level concepts. With high-level concepts in programming, we denote concepts that refer to concepts that are more abstract and removed from detailed operations. Examples of this are design and conventions of larger scripts or Object-Oriented Programming, a way of programming that is quite different from the 'standard' functional programming. This might be due to the subjects stepping away from the very structured way in which they have been taught and trained up until now. Selby refers to multiple ideas of dealing with this. For example, Butler and Morgan (2007) suggested that part of the issue is the mismatch between the difficulty of certain topics and the amount of feedback that is ever given on it. Sakhnini and Hazzan (2008) suggest that, since students often think in analogies, these analogies should be challenged by the use of false analogies and that abstract data types and behaviours should first be discussed and taught in theory before being brought into practice.

3.2.3 Conclusive

Table 1 Bloom's Taxonomy of Teaching in the context of programming and computational thinking adapted by Selby (2015, p. 84)

Bloom's Taxonomy	**Teaching Programming**
Evaluation	Evaluate, test
Synthesis	Create programs, algorithm design
Analysis	Abstract, decompose, discriminate
Application	Abstract, decompose, discriminate
Comprehension	Structures, constructs, facts, types
Knowledge	Structures, constructs, facts, types

Selby (2015) plotted the main programming skills and how they would match up with Bloom's original taxonomy (Table 1). This order to go through the programming seems logical and also can be related to the size of the script someone could write with these skills. The use of structures, constructs, facts, and types are all you need in order to create small and very simple tasks. Being able to break down a problem and see what is important within that (abstraction, decomposition, and discrimination) is needed for the next step and for starting to be able to solve simple problems. Algorithm design is then the next step and understanding this, in combination with the previous

levels, allows someone to start working with complex problems and tasks. Evaluation and testing are then last because these allow the programmer to not only solve the problems at hand, but also start testing how these solutions fare in terms of resources, usability, and time.

Table 2 The proposed order in teaching and perceived difficulty per topic according to Selby (2015, p. 84)

Proposed order of teaching topics	Perceived difficulty of topics (from easiest to hardest)
Constructs, facts, types	Evaluation
How individual constructs work	Algorithm design
Use of constructs in contrived contexts	Generalisation
Discriminate, decompose, abstract	Abstraction of functionality
Create programs, algorithm design	Abstraction of data
Test, evaluate	Decomposition

For both computational thinking and programming, decomposition is the most difficult skill to learn according to Selby (2015, p. 85). Selby continues to explain that a part of this difficulty comes from the lack of experience or incomplete understanding of the problem from the student (Selby, 2015, p. 85). A problem that arises here is that decomposition as a skill is required to build upon to for abstraction, algorithm design and evaluation (see Table 2). The generalisation skill is more linked to Bloom's application. This is due to the fact that generalisation in this context is something heavily applied. Since it builds on the idea that knowledge from one system, problem, or context can be used to deal with similar problems more easily and effectively in another.

3.3 Gamification

In order to learn or be taught how to program, there is a need for understanding the programming logic. This is very important and can be difficult for people who have not yet been exposed to it. Some ways to help with this are using analogies when discussing certain topics or concepts. And Xinogalos (2016, p. 584) also mentions that based on their results the use of pseudo-language can be very welcome and helpful to establish the logic and 'steps' that are used with code, without immediately burdening the students with the pressure of a perfect syntax. This is especially the case when first introduced to programming. With pseudo-language the idea is that we create a structure that is not according to the syntax, but easier to understand for people. We use different terms and a form more sentences, than actual code. This way students can first learn how the code functions and what to keep in mind, creating a greater understanding, before having to write the code in the proper format. Say for example someone wants to apply the Pythagorean theorem in Python. But in order to understand the process better they would first write it out in pseudo-language. This would

look something like this: First get the lengths of the a and b side. Then square each of these values. Add the squared values together. Get the square root of this value and present it as c. Without having to apply any real code, there is now a logical structure and order of operations for acquiring the length of the longest side for a right triangle.

Another potential tool or method for teaching programming and retaining student focus is gamification of the process. Elskiekh and Butgerit (2017) look at using the concept of gamification when teaching programming. A problem with gamification is that it could make the motivation of the student too much focussed on the game-like elements and not enough on the learning itself. Multiple aspects of this build around competition and pride, using leaderboards or points to indicate progress (Elskiekh & Butgerit, 2017, p. 3-4). However, this way of tracking progress and creating motivation can lead to a mindset that focusses more on the competitiveness of the students, and not so much the actual subjects at hand. These elements would also not be as strong or useful if there was just one singular student, which, with the goal of this course, is a very real possibility. Other elements can however be motivating for a student, like considering certain parts of the course as levels or rewarding badges with certain overcome challenges (Elskiekh & Butgerit, 2017, p. 3-4). Since the subjects of the course will be split up in such a way that the next subject often builds on the previously discussed topics, the idea of using these subjects as levels can be easily implemented and can potentially add to the motivation of the student. Breaking down the course structure can also add to the feeling of accomplishment in the student upon successful completion of a segment. For a practical skill such as programming, it is important to give the people being taught hands-on experience. Not only will this help them understand the practical applications of programming. It will also expose them to making mistakes and how to find and fixe these within their code. “Hands-on activities are considered necessary for acquiring problem solving skills and programming capabilities.” (Xinogalos, 2016, p. 580).

Problem solving is one of the main reasons why we apply programming in the first place. So being able to deconstruct the problem and solve it effectively can be very important. Using active learning techniques for programming results in greater satisfaction, improved learning experience, more motivation, and better student performance. Even if this means more work for the teacher (Berssanette & De Francisco, 2021, p. 213-214). Results from Xinogalos (2016) also showed that nearly all students from the inquiry considered solving exercises to be much or very much important. The results also indicate that getting the answers after the students have tried themselves to solve the exercises is considered to be important too (Xinogalos, 2016, p. 580). The results also indicate that performing activities, such as coding along or making exercises, during lectures helps with attention and creates better possibilities for helping students individually when problems arise

(Xinolagos, 2016, p. 582). This is however something that becomes increasingly more difficult as the classroom grows larger.

A nice way for people to learn how far they've come and to solve problems all of their own is to work on a small project. This allows the students to experience just what kind of trouble and challenges you run into. Important here, especially for beginners, is that there are multiple ways to achieve a certain goal with programming. It should, in my personal opinion, not be the goal yet to write the most optimised and cleanest code possible. This is something that will come and develop when they start to explore and work with programming more. This does not mean that adhering to conventions and trying to keep code clear, clean, and readable is not important. It is just not the main priority. For now, the bar should be set at making a functioning code that achieves the goals or answers the problems and that the producer of the code can understand and replicate.

As with many other skills it is important to keep trying and repeating. In doing something multiple times we can improve our code and see smarter or quicker ways to deal with certain challenges. Some people will need extra repetition and exercises for certain topics. This is quite natural as we are all different. But glancing over these extra needs can lead to larger issues later on, as with programming we often build on the knowledge we already have. This also clashes with the availability of resources on the side of the educator. As the teacher will often not have time or resources to maintain a clear overview of the skill level and speed of all their students.

There is also a certain responsibility when it comes to mindset and conventions when teaching someone programming. Not just what the rules and conventions considering the syntax of the code are, but also things like Open Science principles when writing code and data privacy. These might be considered secondary to the actual programming, but they do form important topics if the students want to apply their programming skills in their fields. And are best taught, or at least introduced, early in the educational process.

3.4 Active Learning

A good way to keep students engaged with a course is employing active learning techniques. There are different active learning methods to improve the information intake and degree of participation of students in courses. Whether a method is suitable depends on the size of the group in question, and on the available time. Since the application and preparation of active learning methods is a time-intensive endeavour, however, time constraints have been reported as the main difficulty faced by educators when applying these principles (Berssanette & De Francisco, 2021, p. 214). Regardless, according to the findings of Bersannette and De Francisco (2021), the time spent on employing active learning principles poses as a fruitful investment int students' understanding, as it significantly improves success rates when learning to write code.

An example of having to consider both the available time and structure of a course for choosing appropriate methods, could be the use of the flipped classroom technique. Simply defined this technique means that “that which is traditionally done in class, is now done at home, and that which is traditionally done as homework is now completed in class” (Bergmann & Sams, 2021, p. 13). However, if there is no time planned for homework or if the type of course does not include it in the scope of the course, like with the adult focussed education of this project, this method will not be properly applicable. Even if it is considered to be one of the most popular and widely used active learning techniques for actual classrooms (Berssanette & De Francisco, 2021, p. 209).

A more suitable active learning method for this research’s course plan is project-based learning. Actively working towards a result within the scope of a course project keeps students engaged and allows them to learn “on the go” as they encounter challenges and figure out their own ways of overcoming them. Importantly, an educator has to be available here as well to help with feedback and the teaching of conventions.

Alternative active learning techniques are peer instruction or student-as-teacher. Technically, these are two methods, but what they add is similar. They encourage the student to reflect on and question why a certain thing is the way it is. By asking a student to share their opinions or explain a concept, for example, these methods stimulate thinking beyond a simple yes or no answer, requiring that arguments and explanations supporting their opinions or definitions are given.

3.5 Using GPT’s with learning to use Programming

With its wide range of functionalities, GPT’s can also be used in learning something new. What is important here is to try and avoid letting the model do things for us that we are trying to learn. It is more an extra tool that can help in a variety of ways. But unless clearly instructed, it will start taking tasks out of our hands. This is a pitfall that GPT models seem to keep returning to.

For learning anything, structure is important. A GPT model can help with this in creating schedules based on our goals, skill levels, and available time. This depends on the information we provide, but it can be a helpful tool to try and schedule any or further education and practice.

3.5.1 GPT’s for computational thinking and Python

Models can help us with some of the topics discussed earlier when it comes to computational thinking. Like with all topics it is important to use the model with the idea of learning new skills and developing and not using the model to solve the problems.

When it comes specifically to Python there are some ways in which a GPT can help us without it creating the exact code we need. Some examples are asking for documentation on a certain function, which then allows for follow-up questions regarding this. We can also provide it with code

that we have written and ask for the model to debug it or to check conventions and efficiency. This can help make sure that any code being made is written properly. By referring directly to code however, it becomes more likely for the model to respond with actual code. To prevent that we must indicate to the model that we do not want it to produce any code unless directly and clearly requested. This way the model will give us feedback that we can then process ourselves, helping in the learning process. Depending on the model, however, it might be 'stubborn' and provide actual code examples anyway.

GPT models can also help in creating small problems, challenges, or projects to be solved using Python. This way it helps come up with ways of acquiring practical experiences and running into issues, which allows for personal improvement. These tasks can be created to ramp up in difficulty, or to focus on a specific point of interest, for example.

3.5.2 Prompt engineering

To get the proper responses you want, you need to use the proper prompts. It is already mentioned that GPT models can write code somewhat reliably, especially simpler code. But this does not teach us anything. And asking for help with a coding related problem will result in the model responding with both a solution and, most times, an example of how the solution would look in code. This very much limits what the problem can teach us. It provides a solution that we might be able to use in somewhat different contexts, think generalisation, but the amount of understanding that could have gained is limited.

By either using custom instructions or incorporating it in the first message of a new chat, we need to inform the model of how we would like to respond. For creating such a starting prompt it is often good practice to include certain factors about yourself as well. Think of adding your personal skill-level, your particular goal, the programs or languages you work with, and most importantly, how you would like the responses to be formatted. Simple additions like, "Do not provide examples of solutions unless specifically asked for", are a great way of getting responses without the answers worked out by the model, but still allowing the user to request the examples should they want to check or compare their own code.

As an example I asked the ChatGPT 3.5 model at the beginning of a chat to not provide any code when questions about Python were asked. After acknowledging my request I asked how Object Oriented Programming works in Python. In accordance with my request the model proceeded to explain the different terms and structure without showing me any actual code. My next request was to expound on methods. And here the model did respond with a code example (https://chatgpt.com/share/f8de4029-f09d-46a5-bd89-6f48629d4b78). With the complexity behind GPT models it is not right to use this example as indicative to all GPT's and all 'conversations' with

these. But it does help illustrate that the output from these models might not always be what we expect it to be.

To deal with this issue, Liffiton et al. (2023) have made their own ‘chatbot’ that leverages openai models to help work around the issue where many GPT’s will give you answers and examples that do the work for you. They did this by including certain standard prompts. Using the openai models means that it is not free to use their chat bot, it is however also not expensive and provides a good programming oriented chatbot that can facilitate without becoming a crutch for the student.

Chapter 4 Course Plan

As mentioned before, this research is accompanied by a course plan that builds on the information, methods, and techniques that are mentioned and discussed in the previous chapter. Since now some of the issues and difficulties associated with teaching a topic such as Python programming have been discussed, it is time to show how that knowledge is applied. If it was easy, or at all possible, to make the objectively perfect course for Python programming, it would already exist. For this course, as any other, choices and decisions were made. And these will be discussed here.

Besides the Python programming, a part of the course also revolves around the use GPT models such as ChatGPT. The implementation of such models in both the teaching and the topics to be taught, will also be discussed alongside the specific Python topics. Make link clearer in previous chapter

This chapter will give a clear image of what the course entails and what 'products' are a part of it. Secondly the topics that are part of the course will be discussed and explained. After that the actual structure of the course will be explained and described. And lastly there will be a section talking about the supporting materials for this course.

4.1 what are/is the product

A teaching plan with supporting materials such as presentations and exercises/assignments with corresponding results and answers. The teaching plan is designed to teach the fundamentals of Python programming to people without any programming experience in such a way that it is meaningful and fulfilling, while also paying mind to certain pedagogical principles of teaching programming. It is not the most important that the literal codes become memorised, but that there is a growing understanding in terms of the logic and understanding of Python code and computers. This way it will be easier for students to understand solutions and suggestions they run into from other sources if and when they start their own projects. It will also help create the understanding that programming can seem a lot more daunting than it has to be. Additionally, there will also be a section discussing the use and application of GPT's in programming and in developing those skills. This part is mostly about the ethics and proper applications of such a powerful tool.

4.2 Topics

Since this course is designed to teach to those with little to no experience it will start at the very beginning. First it is discussed and shown how we can set up the computer in such a way that we can use it for Python programming. It is explained what anaconda and Jupyter are and how they are to be used. After this the first real Python programming topics discussed will be the datatypes that Python is built on and an introduction to the syntax that Python uses. With topics growing more

advanced throughout the course, so too will the syntax become more complicated. That's why there is an effort to clearly and visually introduce these changes as the new topics require them.

After the primitives are discussed, the course will start to focus on starting to work with variables and Pythons' own data structures. From there the topics of flow control and loops will be taught. It is at this point that students can start to tackle some problems with Python code and can start to explore some of the possibilities it provides.

From this basis we move to more in-depth discuss how functions work and what they need. This leads nicely into the subject of making functions, and by extent how to make modules. The installation and importing of modules and packages and how to call their variables, classes and functions is then introduced.

The next step is to learn how to work with the data structure of the computer. Learning to create, find, open, read, write and append files and directories. There will also be a segment on using some Operating System functions to navigate (working) directories and find files. The writing and reading of files is mainly done by using flat text files.

Next is the topic of errors, or exceptions, and how to deal with those. The students have seen errors come by, both through personal mistakes and through examples. But now the subject of how these can be 'caught' and dealt with will be discussed. In this part the students learn to work with a try…except… structure. They learn how to get more information on the occurring errors, how to deal with them if they do occur, and also how to gain more information from them.

The final subject that involves actual coding practices is Object Oriented Programming (OOP). This is a different way of programming, compared to how Python has been discussed so far. OOP works with creating a blueprint of sorts, which can be used to make objects. These objects are limited in what they can do or be used for, based on what we define in the blueprint. In general, this works differently than the 'functional', function based, coding we've done so far. Working with objects also makes it easy to theme them in an archaeological context. This will hopefully help make this way of coding easier to understand for archaeologists.

The final topic will be the topic of using GPT models as a tool while programming. It discusses some of the ways it can help, how it can be incorporated into the workflow, but most importantly also how to use it ethically. It will be introduced and referred to as a possible tool, instead of something that can replace the newly gained skills and knowledge of the students.

As a final challenge and converging point of gained skills, there is also a more complex assignment at the end of the course. This assignment is designed to bring together many of the skills that have been acquired, both in thinking about, and actually writing code. It is made to be somewhat of a challenge, while not surpassing the level of the course by too much. The final assignment will test

not just the knowledge of Python programming, but will also demand a certain amount of computational thinking skills. As such it feels like a fitting challenge that could easily be made into a graded assignment depending on the context in which the course might be given, or taken as a template to build upon for a larger scale project.

4.3 Structure

The course plan follows a certain order/structure of topics since the later topics build on the older ones and examples for the newer ones are easier made if the previous parts are already known.

The course itself is also separated in levels (gamification), to both clearly indicate progress and motivate the students.

The majority of the course focusses on function-oriented programming since this is considered to be easier and is also more suited for introducing the basics. Later on there is also the introduction of OOP (Object-Oriented Programming). This is somewhat more complicated and especially needs a different way of thinking about the problems and goals of the code. To introduce this somewhat more difficult way of programming clear archaeological examples will be used.

The discussion of the use of GPT and the ethics surrounding this are discussed last. This is with the goal of having the students develop their own understanding of Python programming and not using it as a crutch to learn. It will be shown and introduced more in the sense of a tool, rather than it being a replacement of their own skill and knowledge.

The students will test their skills and understanding regularly with tasks that both match the new topics that are being discussed, as well as start to use these in combination with earlier topics. These tasks will be a mix of exercises with the new topics and its syntax, and tasks that challenge more the thinking and logic of the students too. This will create a feeling for how to deal with certain problems and how certain methods can be applied in a more practical way.

4.4 why these supporting materials

Exercises and assignments are a great way for students to try out what they've learned. This not only helps them learn and gain experience with the specific syntax but will also test their ability to think the correct way to solve problems with programming. Additionally, the possibilities to make their own mistakes also allows them to learn from these. This often allows for a quicker understanding and learning process than if students were to follow along with somebody else's code. This is also stimulated by questions asking the student what they think of certain code or to explain topics/terms. In this way they need to think further than just a yes or no answer.

The presentations are somewhat different since these are present to make sure that examples and explanations are clearly visible for all students. These can also serve as 'cheatsheets' or reminders to

be left during the time in which the students practice with assignments. This way they can refer to the very basics, before they must adjust and change things to fit the problems their assignments are posing.

Chapter 5 Discussion

In this study and the creation of the accompanying course plan, certain factors were explored. These mostly related to the challenges of teaching Python programming and computational thinking, and how these are skills that might be relevant and valuable to the archaeological field. In this chapter the findings from the research will be discussed and summarised. This will go along with interpretation of the findings and how they have influenced or been considered in the production of the earlier mentioned and discussed products for the course plan. At the end of this chapter, there is also a section discussing some of the challenges and limitations that occurred for this research, not to be confused with the challenges and issues related to the teaching of the topics specifically, as that will be discussed earlier.

5.1 Findings

The first real finding I'd like to mention is my personal surprise at the lack of Python, or other coding, courses in curricula of archaeology studies in neighbouring countries. Admittedly, coding and programming are not among the topics that will immediately come to mind at the term archaeology. But with how the toolset and possibilities are developing, especially with the rise of artificial intelligence, I had expected it to more of an occurrence.

The literature study revealed that one of the most common issues with learning computational thinking or Python skills, has to do with the understanding of the logic it uses. It can be quite difficult for people to follow and understand this logic as it is often very precise and directed. What can make this issue even more complicated in a teaching environment is that it can vary heavily from person to person to what degree computer logic is understood or easily taught to be understood. This in essence leaves one of the larger issues of the topic very open ended and in need for specific attention per student. One way that has shown to be of use when clarifying and explaining computer logic, is visualisations (Ma, et al., 2011, p. 77-78). Similarly to programming like games for children, visualisations can help in understanding a process without being exposed to, or needing to understand, the code behind it. I would argue that, even though it needs a bit more of an understanding of the logic to begin with, the use of pseudo language can also help in a similar way.

The addition of an interpretation of Bloom's Taxonomy for computational thinking and programming really helped in establishing rough guides for recognising skill levels and identifying which skills are more difficult to learn. This in addition with the order in which topics are best taught creates a backbone that I think can serve as a basis for many related courses or lesson plans. I think especially the inclusion of the computational thinking part is very powerful here, because it is very important

to understand in order to create code, yet still feels somewhat separated from the topic of (Python) coding.

To find an objectively optimal or perfect way of teaching any topic to any audience is nearly, if not fully, impossible. There are a lot of varying methods and techniques for improving teaching and learning. For Python we looked mostly at having a possible rubric and using the methods of gamification and some active learning methods. To which degree these are useful when applied depends on the student but also on the availability of time on the side of the educator. To have to invest so much time will be difficult in any context. And where a very personal supervision environment can be very useful and beneficial for the student, this is, generally, not doable for the educator.

5.2 Course plan

5.2.1 Writing Python code

The main structure of the course topics is divided into levels. Creating easily recognisable 'bits' of course that can help with creating a sense of accomplishment when completing it, but also be useful in case the material will later be used as reference material. The use of Selby's interpretation of Bloom's Taxonomy of Learning for computational thinking (2015) has also been useful in helping identify some common issues that can occur and in the structure of the topics that are being taught. However, with the topics being the very basic concepts of Python, I feel the later stages of this model are not and can not be used fully. This is something, I think, that can be useful further down the path of Python coding.

A main part of learning Python programming is trying things out and working with assignments. The structure of these assignments will shift somewhat as the course progresses. The earlier assignments will clearly hint at, or indicate, which functions and methods need to be used, while later assignments become more 'puzzle-like' and challenge the students to see what is being asked of them and how the code can facilitate that. The assignments will also make the student have to explain their thoughts or answers. This is not only a way to keep engagement with the material higher, but also creates a need for thinking more about how or why certain things or concepts are the way they are.

5.2.2 GPT models

The last topic of the course is about the use of GPT models in learning Python and further development of skills. I think the inclusion of this topic is very important because it is something that has become quite mainstream and been heralded as an easy fix-all inclusion in any field of work. There are quite some possibilities and positives to using these models or learning and developing

skills, especially so in a coding context. GPT models can, in most cases, provide good explanations and even examples of code for the level of someone who is still working with and on the basics of Python. However, in my opinion, the models revert too quickly to trying to 'fix' questions you have with examples, even when asked not to. A simple setting to disallow any code examples in responses could be a useful setting for people who want to learn Python by tackling problems themselves. There is also the pitfall, in all fields, of using GPT models to perform full tasks for us. It is ironically problematic that the models can perform these tasks quite well so often. For the models are also known to 'hallucinate'. This is when GPT models start to create or reform data that makes sense to the model but can be complete nonsense or fully false. Some of these hallucinations can seem fitting with a topic for someone who is not versed in said topic. This is one of the reasons why relying to heavily on these models to performs tasks lead to misinformation and the spread of falsehoods. Not to mention it can lead to plagiarism if relied upon for referencing and accessing existing information and publications. Besides this it is also important to know that any trained model will take any bias in the training data into itself as well. Although, I hope, most companies producing GPT models will try to minimize the influence of bias on their models, it is nearly impossible to fully get rid of all bias. This can range for example from a very American centric view to political conventions and values being subtly absorbed. And it can be difficult to distinguish this bias in responses.

Besides these there are also points of discussion such as data safety and data ownership. It can be hard to find, or to verify, what the owner of a GPT model will do with any information and data it is given to respond to. Will this data be used for further training of the models? Can the privacy and safety of this data be ensured? And what about legislation? From a personal experience I have met people who wanted to incorporate openai's ChatGPT for their company, but weren't allowed to, as it would be considered sharing customer information with a U.S. based company and potentially exposing their data to related American laws as well.

5.3 Challenges

One of the common issues with teaching programming, or other computational skills, is working with, and understanding, the related logic. Just how much this is the case can differ drastically from person to person. This can, in my own experience, partially be traced to the exposure and previous use of digital methods and programs a person has. In order to help people better understand this logic, visualisation and pseudo-language have been suggested. I think these work quite well for this and could maybe be summarized as 'analogy'. In essence, teaching someone with no experience how to write code, is not just teaching one skill. In order to code you need to learn the syntax and terminology of the coding language in question, and also the computational logic behind it. By using analogy to explain the latter we can actually focus on just one of theses topics. In explaining what

the process entails without having to interpret it through code, but rather through visualisations or global descriptions, the focus can be put on the structure and logic behind it all. Understanding this logic will also make the learning of how to apply it with code easier, since there is more of an understanding of what the steps mean and must accomplish.

I think a large part of making a good course is having to find a way between being able to support students, while still being able to generalise the course to be suitable for groups and larger classrooms. This however also demands a certain investment of time and resources. Keeping a students engagement and their interest will help with this. One of the most important skills might be to solve problems. Broad as this seems, it is a skill that everyone will have to learn at some point, but it also allows for people to make their mistakes and find solutions that fit it. Not only is this practical in a course, it is also a scenario that will occur a lot if students decide to keep working with Python more in the future. The goal of the course is to understand the basics of Python and to be able to read the code, also to support this point. There are many solutions available online and in documentation for problems that differ from our own, but can still be used as building blocks to accomplish our goals.

Chapter 6 Conclusion

6.1 Research questions and objectives

The main research question for this thesis was, how best to teach Python in an archaeological context? In order to help answer this question, and produce a related product to it, sub questions were established. The answers to these questions lead to, or support, the answer of the main question.

6.1.1 Why specifically Python?

There are multiple reasons as to why Python would be a suitable scripting language to teach in the archaeological field. These reasons can generally be divided into two categories. These being ease of use and learning, and scope of application.

Python as a scripting language has a syntax that is relatively easy to use and read compared to other scripting or programming languages. The syntax clearly creates a structure that also makes it easer to 'isolate' certain blocks of code when writing or reading it. This compared with the fact that many of the specified structures, such as loops, function defining, try…except, etc., are quite similar, leads to Python being relatively easy to learn and read. Understanding the main functionality of this structure means that for each variation there are only small changes that apply to it, rather than an entirely new structure that needs to be learnt.

The scope of applications of Python is very wide. For many goals or programs there are quicker or more efficient languages available, think R for statistical analysis or C++ for more processor efficient programs, however many of these things can also be done with Python. Especially with the use of third-party packages and a very involved and active community, there are few things that cannot be done with Python.

6.1.2 What can Python add to the archaeological field?

There are not many, if any, ways in which Python can help the archaeological field, in which it could not also be useful for other fields. The main draw of using Python in the archaeological field is that it can be used to automate menial tasks, make tools or plug-ins for existing software or even do some data analysis. With a larger investment of time in improving skills with Python even things like machine learning and advancing analytics would be possible. A way in which Python also helps the field of archaeology is the ability to share packages of Python code online. This would mean that archaeologists can develop tools, or part thereof, that can be imported and implemented by another archaeologist anywhere in the world. Because of this, striving to become better in developing, especially with Open Source principles in mind, can serve the entire archaeological field or the respective sub field.

6.1.3 What are common issues associated with teaching Python?

The biggest commonly occurring issue with learning Python, or many other computational skills, is the understanding of computer logic. Being able to interpret and understand what certain actions do, how the computer 'understands' these, and the result it leads to, can be quite difficult to grasp for people who have no experience or knack for it. The main difficulty, from my point of view, for this is that, if there is no experience of working with computer logic, this skill will be taught by also being taught Python. This then in turn is taught better by understanding the underlying logic. As such I think that separating this for people who have trouble with it from the actual use and syntax of code is the best way of doing this. Good examples of this are using visualisation or pseudo code to explain or show how a process goes, without a need for understanding code that would get to those results. The use of analogies, in whatever variant works, can then also help later with teaching Python to someone who already understands at that point what the code is supposed to 'do' and what processes it should go through.

6.1.4 How useful can GPT models be when learning Python?

With the current developments and popularity of using ai, especially in the way of GPT models, it stands to reason that it can be used for learning new subjects, and especially digital subjects. And, especially for earlier levels of Python, I think most GPT's can be a valuable source of support and information. The effectiveness of the model in learning Python lies, however, in how it used and applied. Asking these models directly for help with a problem or what a solution would look like, is often answered with an example that is a simple copy and paste away from being useable and forgotten. It takes practice to find prompts for the models to not do this. And even if specifically asked, they can sometimes fall back into this behaviour.

Using the models instead to ask for theory, to check code, or for cases that lead to exploration and testing of one's skills, can be a way to help people with gaining more skills and knowledge. In their current state, it feels to me, that GPT models will easily become a crutch for new coders. And it must be said, that for some people, this is enough. More in depth understanding or knowledge is not needed, as long as the provided solution works for their current problem. But for people who want to improve their skill and understanding in these topics, take care of what you ask and how you formulate your prompts.

Besides being very useful and interesting as a tool, also beware that there are a lot of topics to be aware of when using GPT models. Remember that every prompt you send will be stored as data by the company that owns that model. Do you know what will happen with that data? It can be unclear to what degree there are privacy rights attached to the data and consider to what degree you might be exposing it to foreign information laws. Finally, also head the fact that the models are far from

flawless. They can generate false data and play it off as being true without it having any grounds or reference. Especially with more complex or niche topics this risk increases.

6.1.5 How best to teach Python in an archaeological context?

The previous research questions have helped to answer the main research question and aided to create a course plan for the basics of Python scripting. Python being a very versatile scripting language with an easy to read and understand syntax seemed like an ideal language for archaeologists to start with. It can help develop new tools that are currently missing in the field, or help automate tasks that are already present.

For the design of the course the literature studied helped by bringing insights for different parts. The use of a modern rendition of Bloom's Taxonomy of learning, but specifically for computational thinking and programming, has helped in establishing an order in which skills should optimally be taught, what skills are considered to be most difficult to teach, and based on which skills and that presence of which ability levels of students could be gaged. The main issue of computational logic has been discussed, along with the idea to use analogies to help people who have trouble with understanding this can be very effective.

The literature also discusses the use of gamification and active learning techniques to help students be engaged in the course, as well as making them have to think about their answers and solutions to questions and problems.

Due to its accessibility and rise in popularity it also seemed appropriate to discuss the use of GPT models for learning Python. These models can help in learning these skills quite well, if not with some points to keep in mind. It is important to use the models as a tool, not as a crutch. It should not be relied upon too heavily if one wants to develop their own skills and knowledge. It is also very important that some of the dangers of using GPT's are considered too. These range from privacy and data protection, to things like training bias and the occasional hallucinations that might occur.

The course plan in itself is quite basic and directed towards archaeology for analogy and explanation. The assignments with it are designed to not just make someone write code, but also try to grow the understanding of how and why it works the way it does. Especially in combination with the supportive material in this thesis I think it can serve as a solid first step in creating Python-literacy.

6.2 Significance and practical implications

Considering how the use of scripting and programming is becoming more and more accessible and useful in all fields of research, I hope that it will become more widely available and used within archaeology as well. It is my opinion that if the baseline of understanding within the field of archaeology with respect to digital methods rises, so to can the bar of innovation and development.

By this I mean that the more we understand and use these skills and programs, the easier it will be to push what we can use them for.

6.3 Limitations and future recommendations

6.3.1 Limitations

When it comes to limitations, I mostly wish time had allowed me to apply tests and iterations to the course material. Taking feedback and responses from real students or test audiences to help improve the material and detect to what degree the methods discussed work as intended.

Another limitation is that I personally have been teaching different topics related to scripting, programming, and other digital methods, some of which also included Python. And with this experience and working with related topics on other projects, it can become tricky to recognise that not everyone has this base knowledge available and ready. This lead to me having to clearly focus for myself on properly explaining terms that are important to build upon.

6.3.2 Future recommendations

The idea of this research came from a point of view that both recognizes the opportunities that lay in the use of programming to automate the boring stuff (joke intended) and to help make the research life of archaeologists better, but in contrast also notices that it is a skill that is not yet taught as much as one would expect considering the impact and use it has in the world around us.

For future programs it could be the case that they will be developed to build upon the base level that the course from this, or any other, Python plan can provide. Be that changing the structure or methods based on accumulated feedback, or providing more specialised courses build on this course by adding more complex concepts for continuing the development of students. If the level to which archaeologists are capable of working with computers, code, and computational thinking improves, then it also becomes easier to teach them further skills. This will also more easily lead to related knowledge and information being shared and spread.

The course plan and pedagogy from this research aim to show that is possible to learn Python programming to a basic level for archaeologists if the structure is designed to help them overcome some of the common issues. It does however not make anybody following this course a full-fledged programmer or developer. This just serves to provide the basic vocabulary and grammar for people to start reading and copying from others. And in doing so, they'll be able to learn more and more of how programming can help. It is my hope that this thesis has shown that modern methods can help us in defining the past.

References

Bibliography

- Anderson, L. W., & Krathwohl, D. R. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives: complete edition*. Addison Wesley Longman, Inc. http://eduq.info/xmlui/handle/11515/18824
- Apiola, M., & Tedre, M. (2012). New perspectives on the pedagogy of programming in a developing country context. *Computer Science Education*, *22*(3), 285-313. https://doi.org/10.1080/08993408.2012.726871
- Armstrong, P. (2010). *Bloom's Taxonomy*. Vanderbilt University Center for Teaching. https://cft.vanderbilt.edu/guides-sub-pages/blooms-taxonomy/
- Bergmann, J., & Sams, A. (2012). *Flip your classroom: Reach every student in every class every day*. International society for technology in education.
- Bers, M. U. (2019). Coding as another language: A pedagogical approach for teaching computer science in early childhood. *Journal of Computers in Education*, *6*(4), 499-528.
- Berssanette, J. H., & de Francisco, A. C. (2021). Active learning in the context of the teaching/learning of computer programming: A systematic review. *Journal of Information Technology Education. Research*, *20*, 201. https://doi.org/10.28945/4767
- Bird, D., Miranda, L., Vander Linden, M., Robinson, E., Bocinsky, R. K., Nicholson, C., Capriles, J. M., Finley, J. B., Gayo, E. M., Gil, A., d'Alpoim Guedes, J., Hoggarth, J. A., Kay, A., Loftus, E., Lombardo, U., Mackie, M., Palmisano, A., Solheim, S., Kelly, R. L., & Freeman, J. (2022) p3k14c, a synthetic global database of archaeological radiocarbon dates. *Scientific Data*, *9*(1), 27. https://doi.org/10.1038/s41597-022-01118-7
- Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H., & Krathwohl, D. R. (1956). *Taxonomy of educational objectives: The classification of educational goals. Handbook 1: Cognitive domain* (pp. 1103-1133). New York: Longman.
- Butler, M., & Morgan, M. (2007). Learning challenges faced by novice programming students studying high level and low feedback concepts. *Proceedings ascilite Singapore*, *1*(99-107).

- Conklin, J. (2005). [Review of *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives Complete Edition*, by L. W. Anderson, D. Krathwohl, P. Airasian, K. A. Cruikshank, R. E. Mayer, P. Pintrich, J. Raths, & M. C. Wittrock]. *Educational Horizons*, *83*(3), 154–159. http://www.jstor.org/stable/42926529
- Elshiekh, R., & Butgerit, L. (2017). Using gamification to teach students programming concepts. *Open Access Library Journal*, *4*(8), 1-7. https://doi.org/10.4236/oalib.1103803
- Francisci, D. (2021). A Python script for geometric interval classification in QGIS: a useful tool for archaeologists. *Environmental Sciences Proceedings*, *10*(1), 1. https://doi.org/10.3390/environsciproc2021010001
- Gellis, J. J., Smith, C. R., & Foley, R. A. (2022, January 24). *Palaeoanalytics.* GitHub. https://github.com/alan-turing-institute/Palaeoanalytics?tab=readme-ov-file#running-pylithics
- Gellis, J. J., Smith, C. R., & Foley, R. A. (2022). PyLithics: A Python package for stone tool analysis. *Journal of Open Source Software*, *7*(69), 3738. https://doi.org/10.21105/joss.03738
- Kim, A. S., & Ko, A. J. (2017, March). A pedagogical analysis of online coding tutorials. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 321-326). https://doi.org/10.1145/3017680.3017728
- Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005). A study of the difficulties of novice programmers. *Acm sigcse bulletin*, *37*(3), 14-18. https://doi.org/10.1145/1151954.1067453
- Liffiton, M., Sheese, B. E., Savelka, J., & Denny, P. (2023, November). Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research* (pp. 1-11). https://doi.org/10.1145/3631802.3631830
- Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *Acm sigcse bulletin*, *41*(3), 161-165. https://doi.org/10.1145/1595496.1562930

- Ma, L., Ferguson, J., Roper, M., & Wood, M. (2011). Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education*, *21*(1), 57-80. https://doi.org/10.1080/08993408.2011.554722
- Mayer, R. E. (Ed.). (2013). *Teaching and learning computer programming: Multiple research perspectives*. New York: Routledge. https://doi.org/10.4324/9781315044347
- Merriam-Webster. (n.d.). Pedagogy. In *Merriam-Webster.com dictionary*. Retrieved June 13, 2024, from https://www.merriam-webster.com/dictionary/pedagogy
- Nederlandse organisatie voor Wetenschappelijk Onderzoek. (n.d.). *Open Science.* https://www.nwo.nl/en/open-science
- Open Source Initiative. (2024, February 16). *The Open Source Definition.* https://opensource.org/osd
- Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education*, *128*, 365-376. https://doi.org/10.1016/j.compedu.2018.10.005
- Sakhnini, V., & Hazzan, O. (2008). Reducing abstraction in high school computer science education: The case of definition, implementation, and use of abstract data types. *Journal on Educational Resources in Computing (JERIC)*, *8*(2), 1-13. https://doi.org/10.1145/1362787.1362789
- Selby, C. C. (2015). Relationships: computational thinking, pedagogy of programming, and Bloom's Taxonomy. In *Proceedings of the workshop in primary and secondary computing education* (pp. 80-87). https://doi.org/10.1145/2818314.2818315
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717-3725. https://doi.org/10.1098/rsta.2008.0118
- Xinogalos, S. (2016). Designing and deploying programming courses: Strategies, tools, difficulties and pedagogy. *Education and Information Technologies*, *21*, 559-588. https://doi.org/10.1007/s10639-014-9341-9

List of figures

List of Tables

Abstract

The goal of this thesis is to establish a pedagogical framework for teaching Python in an archaeological context and discus how it can help the archaeological field. There is also a course plan for the basics of Python that was designed alongside this research. The thesis also includes a critical discussion of the use of GPT models for learning Python or similar skills.

There are three case studies that are discussed to illustrate the potential that the use of Python can have in the archaeological field. The first of these case studies is about the Python package PyLithics, that can create images and generate data based on illustrations of lithic elements. This is mostly tested on earlier or older illustrations of lithics that lack the relevant data. The second case study concerns the use of Python in QGIS to create a new classification for archaeological data. In essence, using Python to add to the functionality of QGIS for the specific data of the research. This case study is also important because it displays the creation of something useful in Python by using and copying code that already existed. The third and final case study explains how Python was used to help get data from a variety of sources with different formats and make sure that the data could be worked with within a single dataset.

Next there is a focus on literature exploring a variant of Bloom's Taxonomy of learning specifically for computational thinking and writing code. This is useful to create a structure and order of how to teach this topic. More literature discusses different methods and techniques to help overcome common issues of learning programming and Python and also provides ways that can help students remain engaged in the course.

One of the biggest challenges with teaching Python is that students need to understand the logic. A good way that is presented to deal with this is by using analogies, be they visualisations, pseudo language, or something similar. The main idea is that by splitting the learning of the logic from the learning to write code, it can become easier to understand how it functions.

Regarding the use of GPT models in learning Python, there are some viable ways to use it as a tool. It can however become a source of answers instead of a source of understanding. Models like these can often give unwanted code examples to accompany their responses and explanations. Besides these points, it is also important to consider the safety of data provided to GPT models, and to consider that some of the responses they provide can be complete nonsense.