# Coding Fundamentals for PhD Students

Friedrich Geiecke

May 2024

Analogously to the pre-sessional courses in mathematics and statistics at the beginning of many economics, business, and other quantitative social science PhD programmes, the idea of this course is to cover broad fundamentals in coding, albeit more applied. The course is divided into 10 units and runs over 5 days, with each day consisting of around 5 hours separated by lunch and coffee breaks. Topics will be introduced with some slides to discuss theoretical background, but most time will be spent working through code samples in notebooks that participants run on their laptops in parallel. Units also include exercises to further practise the respective topics.

## 1 Before you arrive

While we will briefly review Python on the first day, we will then continue with algorithmic complexity, tabular data, visualisation, etc. given the time constraints of the course. It will not be possible to follow these contents without Python knowledge of data types, conditionals, loops, functions, and classes and object-oriented programming. Unless you already have solid knowledge of Python up to classes and inheritance, please complete the following sections of the OpenCourseWare course CS50P online `https://cs50.harvard.edu/python/2022/` before the start of our course:

- 0. Functions, Variables
- 1. Conditionals
- 2. Loops
- 3. Exceptions
- 8. Object-Oriented Programming (the lecture video of this part is longer; if you are running out of time, you can skip its second half for now)

This set of sections starts from first principles and is sufficient preparation in Python for our course, and it will also be an excellent review if you already have some knowledge of these topics. It should take approximately 20 hours to complete with exercises.

Coding is best studied through exercises and projects. The lecture videos contain many code examples and come with associated Problem Sets 0-3 and 8. Each problem set has multiple exercises which should get successively more challenging to solve. If you find you are short on time, you can leave out some later exercise in each of Problem Sets 0-3 and 8, in particular for Problem Set 8 the first two exercises are sufficient as background for our course. If you are instead looking for further exercises to practise, have a look at this alternative Python course from the University of Helsinki `https://programming-24.mooc.fi/all-exercises`. Try to commit to solving exercises just based on the course materials and getting used to reading the Python documentation `https://docs.python.org/3/`, and to use generative AI tools only to explain solutions after attempting the exercises for a solid amount of time without aid. We will discuss and use generative AI tools such as Copilot and ChatGPT in many ways throughout our course, but once investing in studying the fundamentals up to classes from scratch will be the key foundation for all subsequent topics.

There is no need to install Python at this point as all lecture and problem set code of CS50P can be run fully in the browser as explained here `https://cs50.harvard.edu/python/2022/shorts/visual_studio_code_for_cs50/` using a GitHub account. This GitHub account can be created at `https://github.com/`, with free student benefits at `https://education.github.com/benefits`.

Closer to the start of our course, a setup document will be shared with all registered participants containing discussion on which Python installation we will use, how to create and manage code environments, install the code editor VS Code on your computer and configure it for Python, fundamentals of Git and GitHub, basics of the command-line, etc. If you prefer to run the Python code of Sections 0-3 and 8 and their associated problem sets already locally on your computer rather than online in the browser, feel free to send me an email and I can share installation guidelines earlier.

# 2 Syllabus

## Unit 1: Python and Git/GitHub review

We will begin with a brief review of programming in Python (data types, control flow, functions, classes and inheritance) and some basics of version control with Git/GitHub (commits, push/pull, branches, merges and resolving simple merge conflicts). All code and exercises of the course will be distributed via GitHub such that participants will be able to practise using Git throughout the week as well.

## Unit 2: Algorithmic complexity

A short unit follows that discusses the basics of big O notation and algorithmic complexity, and some simple examples of search and sorting algorithms. An

introduction to this topic is common in many first computer science courses, but knowing about the fundamentals can also be helpful for social scientists to understand why some code in empirical projects or theoretical simulations may take very long to run or may not run in a feasible amount of time at all.

### Unit 3: Tabular data

After a brief introduction to general n-dimensional arrays that are typically used for simulations, we will discuss in detail a key library to work with tabular data in Python. We will study how to read tabular data into so-called data frames, reshape and transform it, how to compute common summary statistics, etc. Data in course units form here onwards will frequently be processed in these data frames.

### Unit 4: Visualisation

Python is also a helpful tool for communicating theoretical or empirical research through highly customisable visualisations. We will discuss how to create a range of static as well as interactive plots, and also briefly how to build dashboards or web applications which can be hosted and shared online, e.g. to illustrate results of research projects.

### Unit 5: Textual data and topics in natural language processing

This is the largest unit of the course and will require around one day. We will begin with fundamentals of loading and processing textual data such as character encoding issues and how to try to resolve them, writing scripts to automate optical character recognition for texts contained in images such as scans, or regular expressions to extract information from texts at scale.

Afterwards we will study selected topics in natural language processing for text. Beginning with tokenisation and dictionaries/word counting, we discuss document embeddings, word embeddings, and probabilistic topic models. Because of the time constraints, some of these can only be discussed very briefly, but we will look at code examples for all topics and many references to further resources will be provided. As it is arguably the most salient current area of natural language processing, our main focus for the remainder of the unit will then be to broadly understand how large language models (LLMs) such as e.g. the GPTs work. This will require a review of some fundamental neural network architectures and their training, basics of the attention mechanism, transformers, and also how LLMs may be aligned, e.g. through supervised fine-tuning and reinforcement learning from human feedback.

## Unit 6: Web scraping

Obtaining data from the web in an automated way can be relevant for research projects in different fields. We will study HyperText Markup Language (HTML) and Cascading Style Sheets, build simple websites, and review some common file formats of the web. We will then discuss basic web scraping of tables and static sites, and also more advanced tools which allow to automate web browsers to scrape data from dynamic sites with forms.

## Unit 7: Web APIs

Many data providers today offer Web APIs (Application Programming Interfaces) to download data already in a structured format. We will study how to access such APIs with Python and discuss three examples in more detail which resemble common use cases of APIs in research: The New York Times Archive API that allows to download historical newspaper data since 1851 up to the lead-paragraph level, an API from one of the large language model providers as an example of how to query those large language models at scale which cannot run locally, and the Wikidata API which allows to automate collection of data behind Wikipedia and which can be helpful in many empirical projects in social sciences and beyond.

## Unit 8: Databases

Since larger amounts of data are often stored in databases, some basic knowledge of this topic is helpful for researchers to access such data. We will therefore continue with an introduction to relational databases and SQL, followed by a brief discussion of modern NoSQL databases. We will manage both local as well as cloud databases such as in BigQuery directly through Python.

## Unit 9: Cloud computing

We will discuss fundamentals of cloud computing and set up virtual machines in the cloud, e.g. browser-based notebook servers or more general instances accessed through the command-line with SSH. These cloud instances can e.g. be used for scientific computing (to solve particularly complex theoretical models, run large regressions in empirical projects, etc.) or for continuous data collection through scraping or APIs.

## Unit 10: Project organisation and further topics

Code in notebook files is very helpful e.g. for courses, but for papers or code in production in industry settings, modules in .py-files are used instead. We will discuss how to modularise code, set up and organise projects, and make computations more easily replicable through storing and sharing the underlying code environments. Furthermore, we will briefly explore the debugging and Git

capabilities of VS Code. Along a few exemplary code snippets from alternative languages such as Matlab, Julia, R, or Java, we will then discuss how the concepts studied in this week transfer to many other languages and allow to relatively easily understand their syntax and use them as well. While slower in some numerical tasks, one main advantage of Python is its extremely broad and developed library ecosystem, in particular in machine learning and natural language processing which are increasingly used across quantitative social science research fields. The course therefore concludes with an outlook to further Python libraries which can be helpful in research (e.g. libraries for symbolic mathematics, solving equations numerically, networks, or machine learning).