"The art of debugging is figuring out what you really told your program to do rather than what you thought you told it to do."
- Andrew Singer [1]

1 Introduction

The best way to save time debugging is to write clean code in the first place. However, no matter how well-written a program is, bugs will always appear. Therefore, an extensive debugging toolkit is essential. In the following text, I will present some of the tools I use for debugging and describe the areas in which I want to improve myself during this course and in general.

2 Different Debugging Environments

Different coding and debugging situations call for different tools. Sometimes it is only a small error in a few lines of code. Sometimes it is an extensive program that stops to work for no apparent reasons. To be able to react quickly to all sorts of problems I mainly use the following tools:

IDE (PyCharm) IDEs are probably the most powerful debugging environments. Integrated debuggers allow you to set breakpoints, catch exceptions and inspect variables. Additionally, they have a lot of comfort features allowing you, for example, to instantly jump to definitions or to refactor code.

Jupyter Notebook When debugging data analytics projects, I find it best to use an interactive Jupyter notebook. It allows you to run your code chunk by chunk and to visualize the data quickly which is, in my opinion, the best way to ensure that your calculations are correct.

Command Line On a server or when you only have to fix a small problem, I use the integrated python module pdb (short for python debugger). It allows one to run a program line by line without much effort. An excellent introduction to pdb can be found here[2].

3 Other Useful Debugging Features

Jupyter Magic Commands [3] In a jupyter notebook, you can use magic commands like **%pdb**, **%timeit** or **%prun** to get insights into the execution time or to run the debugger or profiler. These tools are also great for optimization.

Python Logger [4] The python logger is a superb alternative to the classic print statements everyone uses while debugging. It allows you to control the logging granularity and thereby makes it possible to quickly switch between the debugging version of a program and the regular version without changing it.

4 Areas of Improvement

In the future, I plan to use the more advanced debugging features described in this text more extensively. This includes using a logger instead of classical print statements, using an IDE debugger and optimizing code based on detailed insights from specific tools. Additionally, the consequent use of a Version Control System (VCS) like Git makes it easier to find, for example, newly introduced bugs. All in all, debugging is one of the essential skills of a programmer and thus worth improving.

[1] https://medium.freecodecamp.org/how-to-think-like-a-programmer-lessons-in-problem-solving-d1d8bf1de7d2
[2] https://www.machinelearningplus.com/python/python-debugging/
[3] https://ipython.org/ipython-doc/3/interactive/magics.html
[4] https://docs.python.org/3/library/logging.html