Final thoughts

You can order print and ebook versions of *Think Python 3e* from Bookshop.org and Amazon.

Learning to program is not easy, but if you made it this far, you are off to a good start. Now I have some suggestions for ways you can keep learning and apply what you have learned.

This book is meant to be a general introduction to programming, so we have not focused on specific applications. Depending on your interests, there are any number of areas where you can apply your new skills.

If you are interested in Data Science, there are three books of mine you might like:

- *Think Stats: Exploratory Data Analysis*, O'Reilly Media, 2014.
- *Think Bayes: Bayesian Statistics in Python*, O'Reilly Media, 2021.
- *Think DSP: Digital Signal Processing in Python*, O'Reilly Media, 2016.

If you are interested in physical modeling and complex systems, you might like:

- *Modeling and Simulation in Python: An Introduction for Scientists and Engineers*, No Starch Press, 2023.
- *Think Complexity: Complexity Science and Computational Modeling*, O'Reilly Media, 2018.

These use NumPy, SciPy, pandas, and other Python libraries for data science and scientific computing.

This book tries to find a balance between general principles of programming and details of Python. As a result, it does not include every feature of the Python language. For more about Python, and good advice about how to use it, I recommend *Fluent Python: Clear, Concise, and Effective Programming*, second edition by Luciano Ramalho, O'Reilly Media, 2022.

After an introduction to programming, a common next step is to learn about data structures and algorithms. I have a work in progress on this topic, called *Data Structures and Information Retrieval in Python*. A free electronic version is available from Green Tea Press at https://greenteapress.com.

As you work on more complex programs, you will encounter new challenges. You might find it helpful to review the sections in this book about debugging. In particular, remember the Six R's of debugging from Chapter 12: reading, running, ruminating, rubber-ducking, retreating, and resting.

This book suggests tools to help with debugging, including the `print` and `repr` functions, the `structshape` function in Chapter 11 – and the built-in functions `isinstance`, `hasattr`, and `vars` in Chapter 14.

It also suggests tools for testing programs, including the `assert` statement, the `doctest` module, and the `unittest` module. Including tests in your programs is one of the best ways to prevent and detect errors, and save time debugging.

But the best kind of debugging is the kind you don't have to do. If you use an incremental development process as described in Chapter 6 – and test as you go – you will make fewer errors and find them more quickly when

you do. Also, remember encapsulation and generalization from Chapter 4, which is particularly useful when you are developing code in Jupyter notebooks.

Throughout this book, I've suggested ways to use virtual assistants to help you learn, program, and debug. I hope you are finding these tools useful.

In additional to virtual assistants like ChatGPT, you might also want to use a tool like Copilot that autocompletes code as you type. I did not recommend using these tools, initially, because they can be overwhelming for beginners. But you might want to explore them now.

Using AI tools effectively requires some experimentation and reflection to find a flow that works for you. If you think it's a nuisance to copy code from ChatGPT to Jupyter, you might prefer something like Copilot. But the cognitive work you do to compose a prompt and interpret the response can be as valuable as the code the tool generates, in the same vein as rubber duck debugging.

As you gain programming experience, you might want to explore other development environments. I think Jupyter notebooks are a good place to start, but they are relatively new and not as widely-used as conventional integrated development environments (IDE). For Python, the most popular IDEs include PyCharm and Spyder – and Thonny, which is often recommended for beginners. Other IDEs, like Visual Studio Code and Eclipse, work with other programming languages as well. Or, as a simpler alternative, you can write Python programs using any text editor you like.

As you continue your programming journey, you don't have to go alone! If you live in or near a city, there's a good chance there is a Python user group you can join. These groups are usually friendly to beginners, so don't be afraid. If there is no group near you, you might be able to join events remotely. Also, keep an eye out for regional Python conferences.

One of the best ways to improve your programming skills is to learn another language. If you are interested in statistics and data science, you might want to learn R. But I particularly recommend learning a functional language like Racket or Elixir. Functional programming requires a different kind of thinking, which changes the way you think about programs.

Good luck!

Think Python: 3rd Edition