Part I

Getting Basic Tasks Done

1 Prologue: Preparing to Program

The Preface and To the Student sections describe how and why we structured the book and resources the way we did and how to make the most out of them. In the present chapter, we set the stage for the study of programming as an endeavor and Python as a language. We also describe what software needs to be installed in order to make use of the rest of the book.

1.1 What Is a Program and Why Learn to Program?

A **program** is a set of instructions telling a computer what to do. Every action a computer takes, from making a calculation to displaying a graph, is ultimately controlled by a program that a human being writes. This program consists of a file with commands. The computer reads that file and executes the commands one at a time.

The problem is that the language the computer understands is different from the languages that human beings know. Natural language – the language of people – is incredibly rich and is capable of describing so much more than facts and figures. Computer languages, in contrast, are extremely simple with very limited vocabularies and capabilities. This is because a computer can only do a few things:

- Save values.
- Do calculations.
- Ask if something is true or false.
- Accept input (e.g., from a keyboard) and output (e.g., to a screen).
- Do a task over and over again.

In one sense, everything a computer does – whether sending an email, playing a cat video, or modeling the Earth's climate – is the result of many programmers breaking down whatever complex tasks they want the computer to do into some combination of the above capabilities. So, learning how to program means learning how to break down the task we want to do into (very) simple pieces and how to express those tasks in a language – a programming language – that the computer understands. As an aside, we often talk about **code** or coding when referring to the task of programming. Those terms are another way of referring to the syntax of computer languages and the task of writing programs in those languages, respectively.

For a scientist or an engineer, what is the purpose of learning to program? Few scientists or engineers are interested in becoming software developers, and the foundations of modern science were developed using pen, paper, and the human mind. Newton and Darwin did their

calculations without calculators, let alone computers. Today, great science and engineering work can still be done without needing to program a computer. For work involving small datasets or analytical mathematical solutions, pen and paper (and a calculator or spreadsheet) is often enough.

Today, however, we enjoy more data than our predecessors would have believed possible: measurements from satellites, in-situ sensors, or large-scale experiments; models of fluid flow, structures, or biological systems. A spreadsheet is often inadequate to deal with such large datasets. At the same time, the sciences and engineering have been the recipients of an explosion in computational tools for calculations of all kinds. Whether we are looking for a traditional statistical analysis **routine** or want to implement the latest machine learning algorithm, someone else has written a tool we can use. The software engineering community has also developed tools and legal frameworks that enable computational tools to be easily shared and integrated into anyone's programs. The result is that a person who knows how to program can do more science and engineering.

A picture is worth a thousand words, so consider Figure 1.1: This is an image showing the light received by the National Aeronautics and Space Administration (NASA) Kepler Mission's spacecraft (a space telescope designed to search for exoplanets) from the star

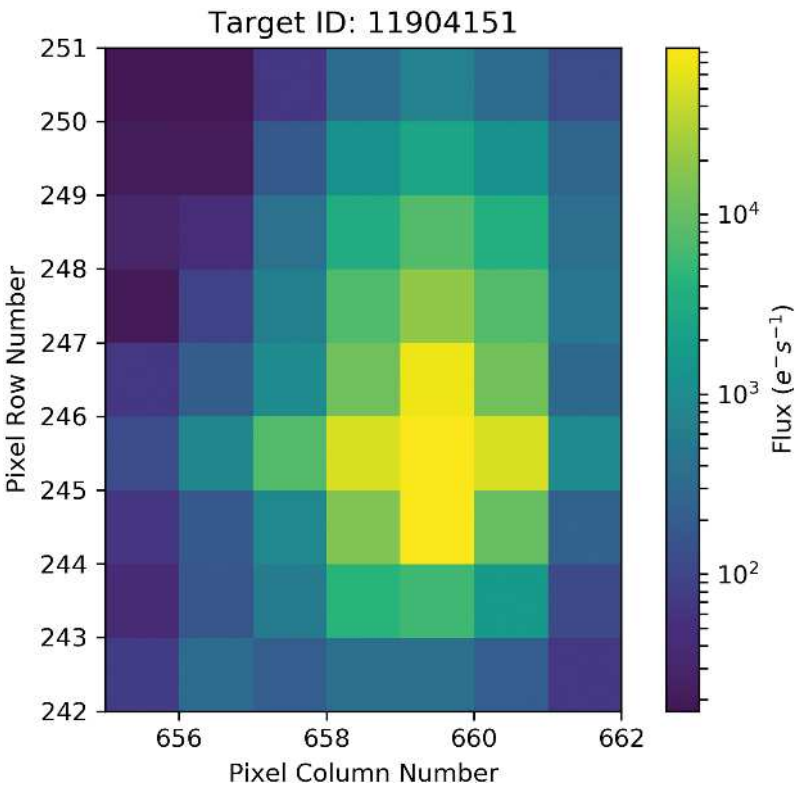

Figure 1.1 Light flux from Kepler-10 during quarter 4.

Kepler-10, around which orbits the first rocky exoplanet discovered by the spacecraft (Kepler-10b).[1] The Python code to read in the data via the Internet from the Mikulski Archive for Space Telescopes (MAST), where the archived data resides, and create the plot is as follows:

```
from lightkurve import KeplerTargetPixelFile
data = KeplerTargetPixelFile.from_archive('Kepler-10', quarter=4)
data.plot(scale='log')
```

That's it! Three lines of code does it! This is why scientists and engineers need to learn to program. We will not unpack these lines of code right now, but after going through the book, we will have the tools to understand their meaning and use.

1.2 What Is Python and Why Learn This Language?

There are many programming languages, and every language has its own strengths and weaknesses. But it is difficult to learn programming in the abstract. To learn programming, we have to use some particular language. Once we learn one language, other languages are more easily learned, but we have to start somewhere. The language we will use in the present work is Python.

While there is no perfect programming language, the Python language is extremely powerful and versatile while at the same time also easily understood and learned. Code written in Python is very clear, so clear that it almost reads like naturally spoken English. For scientists and engineers, Python has capabilities that make it useful for analyzing data and solving mathematically based problems. There is a large community of scientific Python users and developers that are constantly adding to the capabilities of the language to do science. As we go through the text, we will see these characteristics and features of Python. For now, just trust us that Python is a good place to start.

There is, however, one additional reason for learning Python: it is free. Python is an **open source** programming language, that is, a programming language whose underlying code is itself available and open to anyone to examine and use for their own purposes. Python is free in the sense of "freedom." Python is also free in the sense of "no cost." As a result, if we have a connection to the Internet, we can obtain a copy of everything needed to run Python programs. We can use the Python language for our own programs without worrying about whether we can purchase a (possibly) expensive license for a specialized data analysis language or whether a program we write today will be able to be run tomorrow if we distribute it to others or move to another computer ourselves. After learning Python, we can be confident

[1] This example is slightly altered from the one given in the documentation of the lightkurve package (see Vinícius et al. (2018)), a Python package used to access data from the Kepler Mission. That example is at http://lightkurve.keplerscience.org/tutorials/2.02-recover-a-planet.html (accessed August 15, 2018). Information about the Kepler Mission is at www.nasa.gov/mission_pages/kepler/overview/index.html (accessed August 20, 2018).

(as humanly possible) that we will be able to use the language throughout our lives for whatever scientific or engineering purpose – or business, artistic, literary, etc., purpose – we wish.

1.3 Software We Will Need

In this book, we will learn to write Python programs, but in order to get our computer to **run** (or execute the commands of) our programs, we need a little more than the program instruction files themselves. Python most usefully comes as part of a **distribution** of utility programs and tools.

We have written this text assuming version 3.x (e.g., 3.6, etc.) of Python has been installed, through a recent installation of the Anaconda distribution.[2] Installing the Anaconda distribution is preferable to installing Python from the Python Software Foundation's website, or to rely on the version of Python that is included with some operating systems, because the Anaconda distribution includes packages and utility programs that are essential for scientists and engineers.

The Anaconda distribution download page contains detailed directions on how to download and install the software. We provide an up-to-date link to the page at www.cambridge.org/core/resources/pythonforscientists/refs/, ref. 1. This edition is free to individuals and can be installed without administrator privileges. It can be installed on the same computer or account at the same time as another installation of Python and is available for all major operating systems (Windows, Mac OS X, and Linux).

When installed out-of-the-box, the Anaconda distribution provides most of what we need. Through the distribution's tools, we can also add additional utility programs and libraries (called packages) to the distribution, as desired. Use Anaconda Navigator to install additional packages and manage packages. Up-to-date links to the Navigator documentation and getting started guide are at www.cambridge.org/core/resources/pythonforscientists/refs/, refs. 2 and 3 respectively.

Here is a list of additional packages that are used in the current text that might not come with the default installation of the Anaconda distribution. In parentheses, we note which chapters or parts of these packages are first and/or mainly used:

- cartopy (Chapter 14),
- line-profiler (Part IV),
- memory-profiler (Part IV),
- netCDF4 (Chapter 18),
- numba (Part IV),
- pytest (Section 23.3.2).

[2] Most of the code in this book, particularly in the earlier chapters, will also work with Python 2.7.x. However, although there is some scientific and engineering legacy code still written in Python 2.7.x, all major packages have migrated to version 3.x.

To run the code in the sections listed above, please add these packages. Note the line-profiler and memory-profiler packages are also referred to as line_profiler and memory_profiler, using underscores instead of hyphens.

If any of the packages above are already installed, Anaconda Navigator will tell us when we try to install it, so we do not have to worry about overwriting anything. If we do not know whether we have all the packages we need, no worries. If while we are running a Python program we receive a message such as:

```
ModuleNotFoundError: No module named 'netCDF4'
```

we can fix that by installing the package or module named, and then rerun our program.

That should be it for the preliminaries. We are ready now to begin our journey in learning Python programming. We start with using Python to fulfill that basic computational need of a scientist or engineer, the need to have a good calculator.