1
Introduction

1.1 About This Book

This book aims to provide a resource for students, teachers and researchers in chemistry who want to use Python in their work. Over the last 10 years, the Python programming language has been widely adopted by scientists, who appreciate its expressive syntax, gentle learning curve and numerous packages and libraries which facilitate numerical work.

The book is composed of relatively short chapters, each with a specific job. Mostly, these jobs fall into one of two categories: to act as a tutorial on a specific part of the Python language or one of its libraries, or to demonstrate the application of Python to a particular concept in chemistry. For students and teachers, these example applications are chosen to go beyond what can be reasonably achieved with a pencil, paper and a calculator: A brief overview of the chemical concepts is usually given, but there is no in-depth tutorial on these topics. Rather, it is assumed that the reader has some familiarity with the topic and wishes to use Python to solve larger or more complex problems than those usually presented in textbooks. For example, the chapter on Hückel molecular orbital theory (Chapter 33) outlines the assumptions behind this approach to modeling the electronic structure of organic molecules and then demonstrates the use of Python in determining the π molecular orbitals of benzene, which (using an unsymmetrized basis) involves a 6×6 matrix determinant: not a problem to be solved by hand.

Researchers are also increasingly using Python in their work as a computational tool, to manage and transform data, to produce publication-quality figures and visualizations, and even (using the JupyterLab package) as a replacement for laboratory notebooks, and to disseminate data and reproducible analysis. Indeed, Jupyter Notebook was listed in an article in the journal *Nature* as one of the ten computer codes that have transformed science.[1]

[1] J. M. Perkel, *Nature* **589**, 344–348 (2021).

https://doi.org/10.1017/9781009106696.001 Published online by Cambridge University Press

Creating high-quality reports requires some knowledge of Markdown and LaTeX to write equations, tables, and chemical reactions: These are described in Chapter 14. There are further chapters on chemical databases and formats, peak-finding, linear and nonlinear least-squares fitting and symbolic computing with SymPy.

The examples are provided on the website https://scipython.com/chem/ in the form of downloadable Jupyter Notebooks (see Chapter 13), and are supplemented by some exercises (with solutions at the back of the book).

1.2 About Python

Python is a powerful, general-purpose programming language that is well-suited to many of the tasks of scientific computing. It is a "high-level language" in that the programmer does not have to manage the fundamental operations of data type declarations, memory management and so on. In contrast to languages such as C and Fortran, for which the user must pass their code through a "compiler" to generate executable machine code before it is executed, Python programs are compiled automatically into "bytecode" (a kind of intermediate representation between its source and the machine code executable by the processor) by the Python interpreter. This makes the process of code development much quicker: There is a single step to code execution, and any errors are reported in messages returned are generally helpful and specific.

Python and its associated libraries are free and open source, in contrast to comparable commercial languages such as Mathematica and MATLAB. It is available for every major computer operating system, including Windows, Unix, Linux and macOS. It is a highly modular language: A core functionality is provided with the Python distribution itself, but there is a large number of additional modules and packages that extend its functionality. The most notable of these, in the context of scientific computing, are as follows:[2]

- NumPy: a package implementing mathematical algorithms for fast numerical computing, including support for vectors, matrices and multi-dimensional arrays – see Chapters 9 and 18.
- SciPy: a library of scientific computing algorithms for optimization, root-finding, linear algebra, integration, interpolation, signal processing and the numerical solution of ordinary differential equations – see Chapters 21, 22 and 25.
- Matplotlib: a package for visualizing and plotting data, with the ability to generate high-resolution, publication-quality graphs and charts – see Chapter 10.

[2] These packages are sometimes collectively referred to as the Python scientific computing "stack."

- pandas: a library providing high-level data structures for manipulating tabular data (`DataFrames` and `Series`), popular with data scientists – see Chapter 31.
- SymPy: a library for symbolic computation, with support for arithmetic, algebra and calculus – see Chapter 35.
- Jupyter: a suite of applications comprising a platform for interactive computing allowing scientists to share code and data analysis in a way that promotes reproducibility and collaboration – see Chapter 13.

However, Python programs will generally not execute as fast as those written in compiled languages: for heavily numerical work, even Python code using the NumPy and SciPy libraries (which call pre-compiled C routines from Python) will not run as fast as code written in, for example, C, C++ or Fortran. It is also hard to obfuscate the source code of a Python program: to some extent, an open-source philosophy is built-in to the Python ecosystem.

1.3 Installing Python

The official website of Python, `www.python.org`, contains full and easy-to-follow instructions for downloading Python. However, there are several full distributions which include the NumPy, SciPy and Matplotlib libraries to save you from having to download and install these yourself:

- *Anaconda* is available for free (including for commercial use) from `www.anaconda.com/distribution` This distribution includes its own well-documented package manager that can be used to install additional packages, either using a dedicated application or the command-line `conda` command.
- *Enthought Deployment Manager (EDM)* is a similar distribution with a free version and various tiers of paid-for versions including technical support and development software. It can be downloaded from `https://assets.enthought.com/downloads/`.

In most cases, one of these distributions should be all you need. There are some platform-specific notes below.

The source code (and binaries for some platforms) for the NumPy, SciPy, Matplotlib, pandas, SymPy and Jupyter packages are available separately at:

- NumPy: `https://github.com/numpy/numpy`
- SciPy: `https://github.com/scipy/scipy`
- Matplotlib: `https://matplotlib.org/users/installing.html`
- pandas: `https://pandas.pydata.org/`
- SymPy: `www.sympy.org/`
- Jupyter Notebook and JupyterLab: `https://jupyter.org/`

1.3.1 Windows

Windows users have a couple of further options for installing Python and its libraries: *Python(x,y)* (`https://python-xy.github.io`) and *WinPython* (`https://winpython.github.io/`). Both are free.

1.3.2 macOS

macOS, being based on Unix, comes with Python, usually an older version of Python 3 accessible from the Terminal application as `python3`. You must not delete or modify this installation (it's needed by the operating system), but you can follow the instructions above for obtaining a distribution with a more recent version of Python 3. macOS does not have a native *package manager* (an application for managing and installing software), but the two popular third-party package managers, Homebrew (`https://brew.sh/`) and MacPorts (`www.macports.org`), can both supply the latest version of Python 3 and its packages if you prefer this option.

1.3.3 Linux

Almost all Linux distributions these days come with Python 3 but the Anaconda and Enthought distributions both have versions for Linux. Most Linux distributions come with their own software package managers (e.g., `apt` in Debian and `rpm` for RedHat). These can be used to install more recent versions of Python 3 and its libraries, though finding the necessary package repositories may take some research on the Internet. Be careful not to replace or modify your system installation as other applications may depend on it.

1.4 Code Editors

Although Python code can be successfully written in any text editor, most programmers favor one with syntax highlighting and the possibility to define macros to speed up repetitive tasks. Popular choices include:

- Visual Studio Code, a popular, free and open-source editor developed by Microsoft for Windows, Linux and macOS;
- Sublime Text, a commercial editor with per-user licensing and a free-evaluation option;
- Vim, a widely used, cross-platform keyboard-based editor with a steep learning curve but powerful features; the more basic vi editor is installed on almost all Linux and Unix operating systems;

- Emacs, a popular alternative to Vim;
- Notepad++, a free Windows-only editor;
- SciTE, a fast, lightweight source code editor;
- Atom, another free, open-source, cross-platform editor.

Beyond simple editors, there are fully featured integrated development environments (IDEs) that also provide debugging, code-execution, code-completion and access to operating-system commands and services. Here are some of the options available:

- Eclipse with the PyDev plugin, a popular free IDE (`www.eclipse.org/ide/`);
- JupyterLab, an open-source browser-based IDE for data science and other applications in Python (`https://jupyter.org/`);
- PyCharm, a cross-platform IDE with commercial and free editions (`www.jetbrains.com/pycharm/`);
- PythonAnywhere, an online Python environment with free and paid-for options (`www.pythonanywhere.com/`);
- Spyder, an open-source IDE for scientific programming in Python, which integrates NumPy, SciPy, Matplotlib and IPython (`www.spyder-ide.org/`).

The short-code examples given in this book be in the form of an interactive Python session: commands typed at a prompt (indicated by `In [x]:`) will produce the indicated output (usually preceded by the prompt `Out [x]:`). It should be possible to duplicate these commands in a Jupyter Notebook or IPython session (e.g., within the interactive programming environments provided by the Anaconda distribution).