Preface

The Python-Based Laboratory: A Hands-On Guide for Scientists and Engineers provides a learn-by-doing approach to acquiring the Python programming skills needed to implement computer-controlled experimental work. This book is not a manual-like presentation of Python. Rather, *The Python-Based Laboratory* leads its readers to mastery of the popular, open-source Python computer language in its role as a powerful laboratory tool by carrying out interesting and relevant projects that explore the acquisition, production, analysis, and presentation of digitized waveforms. Readers, who are assumed to have no prior computer programming or Python background, begin writing meaningful programs in the first few pages.

The Python-Based Laboratory can be used as a textbook for science and engineering instructional laboratory students who are being taught up-to-date Python-based experimental skills. The book also works well as a self-study guide for professional laboratory researchers, industrial engineers, hobbyists, and electronics enthusiasts seeking to automate tasks using Python. Topics covered include the control of data-acquisition devices (including multifunction data-acquisition hardware and IEEE 488.2-interfaced stand-alone instruments), data file storage and presentation, digitized data concepts (such as resolution, sampling frequency, and aliasing), and data analysis techniques (curve fitting and fast Fourier transform). As readers work their way through the book, they build several computer-based instruments, including a DC voltmeter, digital oscilloscope, DC voltage source, waveform generator, blinking LED array, digital thermometer, spectrum analyzer, and frequency meter. Each chapter concludes with a Do It Yourself project and a Use It! example as well as a healthy selection of homework-style problems, allowing readers to test their understanding and further develop their Python-based experimentation skills.

Python has become a very popular programming choice for laboratory and industrial automation in recent years, often replacing the use of the industry-standard languages LabVIEW (product of NI, formerly, National Instruments) and MATLAB (product of MathWorks). Reasons for Python's growth in this area include the following:

- Python is free and open source, in contrast to the commercial products LabVIEW and MATLAB, making it an attractive option in academic, research, and industrial laboratories.
- Python is widely adopted in education. Because it is relatively easy to learn and transfers well to other domains such as scientific modeling, web development, and data science, it is not unusual for today's students to be taught and then use Python in one or more of their courses.
- Python has a large and active user community, which has developed an abundance of software libraries to carry out standard computing tasks needed in laboratory applications.
- Python software drivers are increasingly being provided by equipment manufacturers for their instruments.

The Python-Based Laboratory takes advantage of Python add-ons (called packages or libraries). Some of these Python add-ons facilitate graphical user interfaces (GUIs) and data presentation, one of the strengths of LabVIEW programming, while other Python add-ons enable

array-based mathematical calculations, a powerful feature of MATLAB. By utilizing these add-ons in *The Python-Based Laboratory*, readers will be able to write Python programs that incorporate the favorable features of both LabVIEW and MATLAB programming.

To serve the largest possible audience, *The Python-Based Laboratory* employs only free and open-source software, namely, Python itself, the IDLE Integrated Development Environment (for program development and execution), the Matplotlib library (for plotting), the Tkinter package (for creating GUIs), the NumPy and SciPy packages (for mathematical functions and array-based mathematical operations), and the PyVISA and NI-VISA packages (for IEEE 488.2 control of stand-alone instruments). Likewise, only data-acquisition (DAQ) devices with free Python driver software are used, specifically, NI DAQ devices with the NI-DAQmx driver and nidaqmx Python package and Measurement Computing Corporation (MCC) DAQ devices with the mcculw package.

The progression of topics in *The Python-Based Laboratory* is as follows:

Chapter 1: Python Program Development. This chapter focuses on writing a simple program to test the parity (i.e., evenness or oddness) of an integer. Central features of Python are presented, including its programming environment (terminal session and program editor), data types, conditional branching, exception (error) handling, and program documentation. In addition, Python's support of object-oriented programming is introduced.

Chapter 2: Graphical User Interface Using Tkinter. Readers use Tkinter, the graphical user interface toolkit included as part of the Python Standard Library, to equip their integer-parity-testing program with a GUI. In the process of creating this program, Python's support of both object-oriented programming (via classes) and functional programming (via functions) is discussed and implemented. Readers carry out the procedure for creating their own custom-made function.

Chapter 3: Counted Looping and Waveform Plots. The properties of a digitized waveform are studied as readers write a sine-wave plotting program. Plotting of the digitized sine-wave data is facilitated by Python's Matplotlib library, while the digitized data are generated by three counted-looping methods—for loop, list comprehension, and vectorization. The latter method is realized through the array-based mathematics of the NumPy library. Python's iterable data structures called list and string are discussed, along with the various ways prewritten software can be imported into a Python program. Finally, the procedure for placing a Matplotlib plot on a GUI is implemented.

Chapter 4: Conditional Looping and Real-Time Plots. Readers learn to use Python's conditional looping structure— the while Loop—to create a real-time plot of a sine wave. The advantage of using Matplotlib's animation module, along with its blitting feature, for such plots is explored and the method for placing an animation on a GUI is executed. Indexing and slicing of iterable data structures is explained and the tuple data structure is defined.

Chapter 5: GUI-Equipped Waveform Simulator. Readers write a digitized waveform simulation program with advanced GUI features, including numeric input controls, a pull-down menu, and an embedded waveform plot. The completed program creates waveforms with sine, cosine, DC level, square, sawtooth, triangle, and user-defined shapes. In writing this program, three Python "utility" modules are created to carry out the following tasks: creation of waveform data with a desired shape, creation of a graphical user interface with interactive controls, and creation of a GUI-embedded plot to display waveform data. These three modules are used in programs created throughout the remainder of the book.

Chapter 6: Introduction to Data-Acquisition Device Features. This chapter assumes that readers have access to a multifunction DAQ device from either NI or MCC. Both NI and MCC are manufacturers of popular, high-quality DAQ devices, where MCC devices are the lower priced option with (satisfactorily) scaled-down specifications. Using the interactive program appropriate for their device (MAX for NI, DAQami for MCC), readers familiarize themselves with the capabilities of a multifunction DAQ device, including analog-to-digital conversion, digital-to-analog conversion, digital input/output operations, and digital pulse counting. In the course of this work, explanations are given of the following important digitized data concepts: analog input modes, range, resolution, sampling frequency, aliasing, digital input/output levels, and digital triggering. The NI USB-6002 and MCC USB-202 DAQ devices are highlighted, but the programs developed apply to most other NI and MCC multifunction DAQ devices.

Chapter 7: Data Acquisition Using NI DAQ Device. After their Chapter 6 orientation to the NI DAQ device's capabilities via MAX, readers write Python programs that execute analog-to-digital, digital-to-analog, and digital input/output tasks on the NI DAQ device as they construct several computer-based instruments, including a DC voltmeter, digital oscilloscope, DC voltage source, waveform generator, and blinking LED array. The necessity for placing GUI and DAQ operations on separate execution threads is explored as well as the need for triggering in an oscilloscope application.

Chapter 8: Data Acquisition Using MCC DAQ Device. After their Chapter 6 orientation to the MCC DAQ device's capabilities via DAQami, readers write Python programs that execute analog-to-digital, digital-to-analog, and digital input/output tasks on the MCC DAQ device as they construct several computer-based instruments, including a DC voltmeter, digital oscilloscope, DC voltage source, waveform generator (for upgraded MCC models), and blinking LED array. The necessity for placing GUI and DAQ operations on separate execution threads is explored as well as the need for triggering in an oscilloscope application.

Chapter 9: Data Files and Character Strings. Readers learn how to use NumPy functions to store data in, and read data from, one- and two-dimensional spreadsheet-formatted computer files. Numeric formatting, escape sequences, and the Unicode character coding scheme are covered. The chapter concludes by giving the NumPy-based method for writing and reading binary-formatted files.

Chapter 10: Data Analysis: Curve Fitting. A thermistor is a semiconducting device commonly used as a temperature sensor. A physical model for a thermistor's temperature-dependent resistance is presented and then readers write GUI-equipped Python programs that fit a thermistor's temperature versus resistance calibration data to various functional forms consistent with the physical model. First, the NumPy polynomial curve-fitting function is used to fit the data to a third-order polynomial. Next, SciPy's more general nonlinear fitting function is used to fit the data to several alternate functions, including the widely used Steinhart–Hart equation. Goodness of the fitted curve is judged via residual plots. After obtaining the fitted calibration curve for a thermistor, a computer-based digital thermometer is constructed.

Chapter 11: Data Analysis: Fast Fourier Transform. Using SciPy functions, readers write Python programs to investigate the proper use of the fast Fourier transform (FFT). Readers apply SciPy's complex Fourier transform function to various sinusoidal inputs, demonstrating that this function produces the expected complex-amplitude spectra. In addition, a GUI-equipped program that simply plots the magnitude of the complex-amplitude as a function of frequency is used to explore the effects of spectral leakage,

windowing, and aliasing. With that understanding in place, a computer-based spectrum analyzer is constructed.

Chapter 12: Control of Stand-Alone Instruments Using VISA. Using PyVISA and NI-VISA, Python-based control of an IEEE 488.2-interfaced stand-alone instrument over the Universal Serial Bus (USB) as well as the General Purpose Interface Bus (GPIB) is studied. The Agilent 34410A digital multimeter (which has the same command set as digital multimeters from other manufacturers) is used to demonstrate the central concepts of IEEE 488.2 interface bus communication between a computer and stand-alone instrument, including synchronization methods. Readers write GUI-equipped Python programs to control the instrument's data-taking functions. In the latter portion of the chapter, reader create an instrument driver for the digital multimeter using the Python dictionary data structure as a lookup table for instrument specifications.

Appendix A: Installing Python and IDLE. Instructions are given to assist readers in loading Python and IDLE onto their computer. IDLE is used throughout the book to create and execute Python programs.

Key features of *The Python-Based Laboratory* include its emphasis on real-world problem solving, its early introduction and routine use of data-acquisition hardware, its Do It Yourself projects and Use It! examples at the end of each chapter, and its healthy offering of back-of-the-chapter homework problems.

Real-world problem solving: Chapter topics and exercises provide examples of how commonly encountered problems in the laboratory are solved by scientists and engineers. Python features, along with relevant mathematical background, are introduced in the course of solving these problems. The "best practice" strategies of modularity and code reuse are emphasized so that readers optimize their use of Python. In addition, many of the book's programs are equipped with a graphical user interface, a desired feature in computer programs that control experiments.

Data-acquisition usage throughout: Exercises involving DAQ hardware appear early and then routinely in *The Python-Based Laboratory*. Of particular note, following the book's first five software-only chapters that teach the fundamentals of the Python programming language, data acquisition using a DAQ device is covered in Chapters 6 and, depending on the manufacturer of your DAQ device, 7 or 8. For a professor or self-learner who wishes to devote only three or four weeks to instruction in Python-based data acquisition, Chapters 1 through 7 (or 8) will provide the needed instructional materials. For those planning a more comprehensive study of Python-based lab skills, a digital thermometer and spectrum analyzer are constructed in Chapters 10 and 11, respectively. In Chapter 12, data are acquired remotely from a stand-alone instrument using the USB and/or GPIB interface bus. Additionally, commonly used interfacing circuits consisting of low-cost integrated circuits are presented. Circuits include anti-aliasing filters, an audio amplifier, and a thermocouple signal conditioner.

Do It Yourself projects: Each of the book's chapters concludes with a Do It Yourself project that poses an interesting problem and directs readers in finding a solution by applying the chapter's material. In some chapters, this project involves writing a program that carries out a statistical analysis of flipping coins (Chapter 3), creates a square wave from a combination of sine waves (Chapter 5), or writes and reads a binary-formatted data file (Chapter 9); in other chapters, the reader constructs a computer-based instrument including a millisecond-resolution stopwatch (Chapter 6), digital thermometer (Chapter 10), and a spectrum analyzer (Chapter 11).

Use It! examples: Ready-to-use example programs, which carry out common tasks encountered in laboratory work, are presented at the end of each chapter. Some of these examples involve programming solutions, for example, showing how to document programs, test-run Python coding as "main" programs in order to find programming bugs, and save data during runtime. Other examples are low-cost hardware solutions, including anti-aliasing through the use of an eighth-order Butterworth lowpass filter, amplification and cold-junction compensation for a thermocouple temperature measurement, and configuration of a LAN/Ethernet interface.

Back-of-the-chapter homework problems: A selection of homework-style problems is included at the end of each chapter so that interested readers can further develop their Python-based skills. In some of these problems, readers test their understanding by applying the chapter topics to new applications (e.g., lock-in detection); in others, readers use programs written within the chapter to explore important experimental issues (e.g., frequency resolution of a fast Fourier transform). Finally, a number of problems introduce readers to features of Python relevant to, but not included in, the chapter's text (e.g., code-execution timer).

To aid readers in creating their Python programs, the following conventions are used throughout the book:

- *Italic* text signals the first-time use of important terms and concepts and highlights character strings to be entered on a program's graphical user interface using the keyboard.
- **Bold** text designates pull-down menu selections and graphical user interface label names.
- `Constant Width` text is used for the Python programs to be written by readers as well as within paragraphs to refer to Python program elements. The programs themselves are within a gray background.
- **Straight Bold** text designates the names of programs written by readers as well as the path within the computer file system to those programs.

Any suggestions or corrections are gladly welcomed and can be sent to John Essick at jessick@reed.edu.

Updates, answers to frequently asked questions, and ancillary materials for *The Python-Based Laboratory* (including Python code for all programs within the book as well as solutions to the problems) are available at http://academic.reed.edu/physics/faculty/essick.

Additionally, the book's Python programs and problem solutions can be downloaded at http://www.oup.com/us/essick.

I'd like to express my gratitude to the Physics Department at the University of Newcastle, Australia who inspired me to undertake this project through a Fulbright Specialist visit, especially to Vicki Keast for arranging my visit and Lachlan Rogers for sharing his deep experience with Python in teaching and in research. Also, much thanks to David Horn for his support with Digilent products and many thanks to Devon Essick and Jennifer Heath for reading portions of the book's manuscript and providing invaluable feedback.

For their support and assistance in preparing *The Python-Based Laboratory*, I thank Dan Taber and David Lipp of Oxford University Press and Thilagavathi Ezhumalai of Integra Software Services. For their helpful comments and suggestions, I express my appreciation to the four anonymous reviewers who provided early comments on the book as part of the OUP peer review process.

Finally, to my wife Katie: Thank you for your love and support while I worked on this project.

John Essick
Portland, Oregon

Contents