

# Streamlining Your Research Laboratory with Python

Mark F. Russo • William Neil



**WILEY**



## **Streamlining Your Research Laboratory with Python**



# Streamlining Your Research Laboratory with Python

*Mark F. Russo, PhD*

*The College of New Jersey  
Ewing, NJ, USA*

*William Neil*

*Bristol Myers Squibb Company  
Princeton, NJ, USA*

**WILEY**

Copyright © 2025 by Mark F. Russo. All rights reserved, including rights for text and data mining and training of artificial intelligence technologies or similar technologies.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.

Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at [www.copyright.com](http://www.copyright.com). Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permission>.

The manufacturer's authorized representative according to the EU General Product Safety Regulation is Wiley-VCH GmbH, Boschstr. 12, 69469 Weinheim, Germany, e-mail: [Product\\_Safety@wiley.com](mailto:Product_Safety@wiley.com).

Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Further, readers should be aware that websites listed in this work may have changed or disappeared between when this work was written and when it is read. Neither the publisher nor authors shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at [www.wiley.com](http://www.wiley.com).

*Library of Congress Cataloging-in-Publication Data:*

Names: Russo, Mark F., author. | Neil, William (Automation specialist), author.  
Title: Streamlining your research laboratory with Python / Mark F. Russo, William Neil.  
Description: Hoboken, New Jersey : Wiley, [2025] | Includes bibliographical references and index.  
Identifiers: LCCN 2024062194 | ISBN 9781394249886 (hardback) | ISBN 9781394249909 (epdf) | ISBN 9781394249893 (epub)  
Subjects: LCSH: Laboratories—Data processing. | Python (Computer program language)  
Classification: LCC Q183.A1 R885 2025 | DDC 502.85/5133—dc23/eng/20250127  
LC record available at <https://lcn.loc.gov/2024062194>

Cover Design: Wiley

Cover Image: Photo by Min Fang, © Designer/Getty Images

Set in 9.5/12.5pt STIXTwoText by Lumina Datamatics

*For my girls, Jean, Emily, and Gabrielle. You three are my world.*

**—Mark**

*For my wife, Cindy, and our sons, Billy, Matthew, and Andrew. I am proud of you and love you.*

**—William**



# Contents

## Preface *xv*

## 1 Introduction *1*

- 1.1 Python Implementations *1*
- 1.2 Installing the Python Toolkit *2*
- 1.3 Python 3 vs. Python 2 *2*
- 1.4 Python Package Index *3*
- 1.5 Programming Editors *5*
- 1.6 Notebook Editors *5*
- 1.7 Using the Jupyter Notebook Interface *7*
- 1.8 JupyterLite *8*
- 1.9 Things Change *9*
- 1.10 Key Takeaways *9*

## 2 Language Basics *11*

- 2.1 Python Interactive Console *11*
- 2.2 Data Types *11*
- 2.3 Variables and Literals *14*
- 2.4 Strings *15*
  - 2.4.1 Simple Strings *15*
  - 2.4.2 Multi-Line Strings *16*
  - 2.4.3 Escape Characters in a String *16*
  - 2.4.4 Raw Strings *18*
  - 2.4.5 Formatted Strings *18*
  - 2.4.6 Strings as Objects *19*
  - 2.4.7 Characters and Encodings *21*
- 2.5 Expressions Using Operators *22*
  - 2.5.1 Arithmetic Operators *22*
  - 2.5.2 Assignment Operators *24*
  - 2.5.3 Comparison Operators *25*
  - 2.5.4 Boolean Operators *26*
  - 2.5.5 Chaining Comparisons *28*

- 2.5.6 Comparing Floating-Point Numbers 29
- 2.6 Functions and How to Use Them 29
  - 2.6.1 Invoking Functions 30
  - 2.6.2 Built-in Functions 31
  - 2.6.3 The `math` Module for Additional Mathematical Functions 31
  - 2.6.4 The `random` Module for Pseudo-Random Number Generation 33
  - 2.6.5 The `time` and `datetime` Modules for Handling Dates and Times 36
  - 2.6.6 The `sys` Module for System Interactions 39
  - 2.6.7 Scope and Namespace 40
- 2.7 Your First Python Program 41
- 2.8 Key Takeaways 42

### **3 Data Structures 45**

- 3.1 Lists 45
  - 3.1.1 Introducing Lists 45
  - 3.1.2 Global Functions That Operate on Lists 46
  - 3.1.3 Accessing List Elements 46
  - 3.1.4 Slicing Lists 47
  - 3.1.5 Lists Operators 49
  - 3.1.6 Lists as Objects 51
- 3.2 Tuples 55
  - 3.2.1 Introducing Tuples 55
- 3.3 Dictionaries 56
  - 3.3.1 Introducing Dictionaries 56
  - 3.3.2 Global Functions That Operate on Dictionaries 57
  - 3.3.3 Accessing Dictionary Items 57
  - 3.3.4 Dictionary Operators 58
  - 3.3.5 Dictionaries as Objects 59
- 3.4 Sets 61
  - 3.4.1 Introducing Sets 61
  - 3.4.2 Global Functions That Operate on Sets 62
  - 3.4.3 Accessing Set Elements 62
  - 3.4.4 Set Operators 62
  - 3.4.5 Sets as Objects 64
- 3.5 Destructuring Assignment 65
- 3.6 Key Takeaways 66

### **4 Controlling the Flow of a Program 67**

- 4.1 Conditional Execution 67
  - 4.1.1 If-Statements 67
  - 4.1.2 If-Else Statements 68
  - 4.1.3 If-Elif-Else Statements 70
  - 4.1.4 If-Statement Strategies 71

4.1.5	Truthy and Falsy Values	74
4.1.6	Conditional Expressions	75
4.2	Repeated Execution	76
4.2.1	While-Statements	76
4.2.2	For-Statements	81
4.2.3	For-Statements with Range	83
4.2.4	Break and Continue	84
4.2.5	Comprehensions	85
4.3	Key Takeaways	87
<b>5</b>	<b>Custom Functions and Exceptions</b>	<b>89</b>
5.1	Defining Custom Functions	89
5.2	Arguments and Parameters	94
5.3	Names and Scope	97
5.3.1	Local vs. Global	97
5.3.2	Built-in and Nonlocal Scope	99
5.4	Scope vs. Namespace	100
5.5	Organizing Your Code with Modules	102
5.6	Decorators	104
5.7	How Things Go Wrong	106
5.8	Python Exceptions	106
5.9	Handling Exceptions	107
5.10	Raising Your Own Exceptions	110
5.11	Key Takeaways	112
<b>6</b>	<b>Regular Expressions</b>	<b>115</b>
6.1	Matching Literal Text	115
6.2	Alternation	116
6.3	Defining and Matching Character Classes	116
6.4	Metaclasses	117
6.5	Pattern Sequences	117
6.6	Repeating Patterns with Quantifiers	118
6.7	Anchors	119
6.8	Capturing Groups	120
6.9	Regular Expressions in Python	122
6.10	Project – A Formula Mass Calculator	123
6.11	Key Takeaways	130
<b>7</b>	<b>Working with Data</b>	<b>131</b>
7.1	A File System Primer	131
7.2	Text Files	133
7.3	Reading and Writing Text Files	134
7.4	Working with Comma-Separated Values (CSV) Files	139

7.5	The <code>csv</code> Module	144
7.6	Reading and Writing Excel Spreadsheet	145
7.6.1	<code>openpyxl</code> Workbook Object	146
7.6.2	<code>openpyxl</code> Worksheet Object	147
7.6.3	<code>openpyxl</code> Cell Object	148
7.7	Project – Generate a Random Sample Layout in a Spreadsheet	148
7.8	Project – Forecast Monthly Sample Processing	150
7.9	Managing the File System	154
7.9.1	The Path Object	155
7.9.2	Path Properties	155
7.9.3	Path Attributes	156
7.9.4	Operating On a Path	157
7.9.5	Combining Paths	158
7.9.6	The <code>shutil</code> Module for High-Level File Operations	159
7.10	Walking a File System Tree	159
7.11	Project – Find Duplicate Files	161
7.12	Working with Zip Files	163
7.12.1	<code>ZipFile</code> Object	163
7.12.2	<code>zipfile</code> Path Object	163
7.12.3	Creating Zip Archives	164
7.13	Working with Standard Data Formats	165
7.13.1	JSON – JavaScript Object Notation	165
7.13.2	<code>json</code> Python Module	166
7.13.3	XML – Extensible Markup Language	168
7.13.4	Python XML modules	170
7.13.5	Other Standard Data Formats	176
7.14	Key Takeaways	176
<b>8</b>	<b>Web Resources</b>	<b>179</b>
8.1	TCP/IP Networks – What You Need to Know	179
8.1.1	Internet Protocol	180
8.1.2	Transmission Control Protocol	180
8.1.3	Connections and Ports	180
8.1.4	Application Layer Protocols	181
8.1.5	IPv4 vs. IPv6 Addresses	181
8.1.6	Proxy Servers	181
8.2	Introduction to Hypertext Transfer Protocol	182
8.2.1	The Uniform Resource Locator	183
8.2.2	Anatomy of an HTTP Request	184
8.2.3	Anatomy of an HTTP Response	186
8.3	Web Services and the Python Requests Module	187
8.3.1	HTTP GET Requests and the Response Object	188
8.3.2	HTTP POST Requests	189

- 8.3.3 Binary Responses 190
- 8.3.4 Customizing the Request Object 193
- 8.3.5 Verifying Certificates and Encryption 193
- 8.3.6 Other requests Module Options 194
- 8.4 Project – Print Weather Forecast for a Location 195
  - 8.4.1 National Weather Service API Web Service 195
  - 8.4.2 Getting Forecast URL from Geolocation 197
  - 8.4.3 Loading and Processing Forecast Data 197
  - 8.4.4 Completed Program to Generate Temperature Forecast 198
- 8.5 Project – Scraping HTML Page Content 201
- 8.6 Key Takeaways 206
  
- 9 Data Analysis and Visualization 209**
  - 9.1 JupyterLab 209
  - 9.2 Scientific Plotting with Matplotlib 211
    - 9.2.1 The pyplot Submodule 212
    - 9.2.2 The pyplot.plot() function 212
    - 9.2.3 Customizing a Plot 213
    - 9.2.4 Multiple Curves on a Single Plot 214
    - 9.2.5 Additional Plot Types 214
    - 9.2.6 Multiple Axes on a Single Figure 214
    - 9.2.7 Other Useful Functions 216
    - 9.2.8 Project – Plotting Weather Forecast 216
    - 9.2.9 Project – A Custom Microplate Heat Map 219
    - 9.2.10 Other Scientific Plotting Libraries 222
  - 9.3 NumPy – Numerical Python 223
    - 9.3.1 Creating ndarray Objects 223
    - 9.3.2 Working with ndarray Objects 223
    - 9.3.3 Accessing and Updating ndarray Elements 224
    - 9.3.4 Broadcasting 226
  - 9.4 pandas DataFrame 226
    - 9.4.1 Creating and Inspecting DataFrames 227
    - 9.4.2 Filtering DataFrames 228
    - 9.4.3 Project – A Screening Experiment 230
  - 9.5 SciPy – A Library for Mathematics, Science, and Engineering 239
    - 9.5.1 Descriptive Statistics with SciPy 239
    - 9.5.2 Hypothesis Testing 239
    - 9.5.3 Project – Running Hypothesis Tests on Two Samples 240
    - 9.5.4 Project – Comparing Liquid Handler Syringe Performance 243
    - 9.5.5 Linear Regression 245
    - 9.5.6 Fitting Nonlinear Models to Data 246
    - 9.5.7 Project – Four-Parameter Logistic Regression 248
  - 9.6 Key Takeaways 253

<b>10</b>	<b>Report Generation</b>	<b>255</b>
10.1	BytesIO Object	255
10.2	Generating Reports in Microsoft Word	257
10.2.1	Document Object	257
10.2.2	Paragraph Object	258
10.2.3	Run Object	259
10.2.4	Picture and InlineShape Objects	260
10.2.5	Table Object	261
10.2.6	Project – Generate a Complete Word Report	263
10.3	Generating Microsoft PowerPoint Presentations	266
10.3.1	Presentation Object	266
10.3.2	Slide Objects	267
10.3.3	SlideShapes Object	268
10.3.4	Length Objects	269
10.3.5	Table Object	269
10.3.6	Project – Generate a PowerPoint Document with Figures and Tables	272
10.4	Generating PDF File Reports	275
10.4.1	ReportLab PDF Generation Process	275
10.4.2	Creating a Canvas Object	276
10.4.3	Setting Canvas Styles	276
10.4.4	Managing Text Blocks with PDFTextObjects	278
10.4.5	Canvas State Stack	281
10.4.6	Drawing Images	281
10.4.7	PLATYPUS for Page Layout	283
10.4.8	Project – Generate a Complete PDF Report	283
10.5	Sending Email Programmatically	287
10.5.1	Simple Mail Transfer Protocol	288
10.5.2	SMTP Mail Server	288
10.5.3	Send a Simple Email Message	288
10.5.4	Sending Email Messages over a Secure Connection	291
10.5.5	Building an Email Message with Attachments	292
10.6	Serving Results with an HTTP Server	294
10.7	Key Takeaways	296
<b>11</b>	<b>Control and Automation</b>	<b>297</b>
11.1	Concurrency in Python	297
11.2	Asynchronous Execution	299
11.3	Concurrent Programs with AsyncIO	300
11.4	Asynchronous Instrument Control and Coordination	304
11.4.1	Project – Integrated Laboratory System Control and Coordination	304
11.5	Communicating over a Serial Port	311
11.5.1	Reading Barcodes from a Serial Port	312
11.5.2	Project – Scanning Sample Tasks into a Running Controller	318

11.6	Execute Remote Commands over HTTP	319
11.6.1	A Basic HTTP Server with <code>aiohttp</code>	320
11.6.2	Routing an HTTP Request to a Custom Python Function	321
11.7	Persistent Network Connections using a WebSocket	325
11.7.1	A User Interface for Asynchronous Networked Programs	326
11.7.2	WebSocketResponse and FileResponse Objects	326
11.7.3	Project – A Browser-Based WebSocket Message Broadcaster	327
11.7.4	Project – A Browser User Interface to Schedule Samples for Analysis	335
11.8	Responding to File System Changes	338
11.8.1	Watching a Directory for Changes with <code>watchfiles</code>	338
11.8.2	File System Monitoring Options	340
11.9	Executing Tasks on a Schedule	341
11.9.1	<code>sched</code> Module	342
11.9.2	Project – Taking and Sending Images on a Schedule	344
11.10	Key Takeaways	348
	<b>Postface</b>	351
	<b>References</b>	353
	<b>Appendix A: ASCII American Standard Code for Information Interchange</b>	357
	<b>Index</b>	359



## Preface

Python [1] is a powerful and versatile programming language that has found widespread adoption in numerous scientific disciplines, including the research laboratory. Its simplicity, readability, and extensive libraries make it an invaluable tool for scientists and researchers who need to process data, conduct data analysis, visualize results, and automate routine tasks. In the context of research laboratories, Python's user-friendly nature, coupled with its vast ecosystem of scientific libraries and tools, can transform the way experiments are automated as well as the way data is collected and analyzed.

Python's appeal lies in the fact that it is at once both accessible to novice programmers and sufficiently powerful for experienced programmers. It boasts a straightforward syntax that minimizes the learning curve, enabling researchers to quickly grasp the essentials of programming and apply them to their specific scientific endeavors. Furthermore, its open-source nature ensures that a vibrant community continuously enhances and extends its capabilities, offering an ever-evolving set of tools tailored to the needs of a broad and diverse community.

We do not assume that the reader has prior knowledge of Python programming. Early chapters of this book provide a thorough introduction to the aspects of Python programming that are critical to its application by researchers, especially in a research laboratory setting. That said, we do not intend to cover Python programming in its entirety. Our focus is on helping researchers streamline their laboratory operations, including experimental data collection, analysis, and reporting.

We survey the broad variety of ways in which Python may be used in a research laboratory. We delve into the various aspects of Python that make it an ideal choice for scientists and researchers. We examine its key features, its role in automation, data collection, data analysis, visualization, and scientific computing, and how it can be integrated seamlessly into laboratory workflows. Additionally, we explore some real-world examples of Python's application in research settings, demonstrating its potential to streamline processes, improve productivity, and foster innovation.

Throughout the book, we develop numerous Python programs to solve real practical problems faced by research scientists. All source code may be downloaded from the book's GitHub repository at <https://github.com/russomf/syrlwp>. All corrections and updates will be found there as well.

Whether you are a biologist analyzing genetic data, a chemist scouting synthesis routes, an engineer optimizing machine parameters, or a social scientist studying human behavior, Python has the flexibility and adaptability to be a powerful and indispensable tool in your research toolkit. Join us on a journey into the world of Python programming for research laboratories. You will gain insights into how this versatile language can empower you to unlock new possibilities, accelerate your research, and contribute to scientific advancements.



# 1

## Introduction

Python is one of the most popular programming languages, and for good reasons. Among the principles that guide the design of the language, called the *Zen of Python*, is the principle that *Readability Counts*. You will discover this repeatedly throughout your learning journey. Unlike source code that you may have encountered in the past written in other programming languages, Python will not appear to the untrained eye as an undiscovered form of hieroglyphics. If written well, Python source code can be relatively easy to read and understand, and it can be equally straightforward to write. Perhaps this is what has driven the popularity of Python.

As a general-purpose programming language, you will find that learning Python provides you with the power to solve virtually any computing problem that you may encounter. This includes data collection and processing, instrument control, scientific computations, publication quality graphing, report generation, and much more. Many of these packages are designed to solve the kinds of scientific problems encountered in a research laboratory. We will cover many of the most popular and widely used scientific Python packages. When combining Python's clean and simple syntax with over 600,000 packages that are freely available in the Python Package Index (PyPI), you will find that you have at your fingertips an incredibly powerful toolkit to solve authentic scientific and research laboratory problems.

Another incredible feature of Python is that it has been selected as one of the two most popular languages used for Data Science. As a research scientist, experimental data is likely the lens through which you learn about the world. While you may never need the full range of advanced numerical, statistical, and modeling features required by a professional data scientist, it is no doubt that you will benefit from the power of Python to operate legitimately in the Data Science field.

Finally, it is worth noting that Python is an open-source language, which means it is free to use and has a large and active community of developers. This community continuously maintains and improves Python's vast array of libraries and other tools, making it a robust platform for scientific research. Python is available for most operating systems, which ensures that you will be able to run it wherever you need it, even on microcontrollers.

### 1.1 Python Implementations

Although not formally standardized like other programming languages such as C, C++, and JavaScript, the Python language syntax is defined by *The Python Language Reference* [1] as well as its reference implementation in C called CPython, which is available for all major operating systems. Both the language reference documentation and CPython implementation are available from Python's official website at <https://www.python.org> [2].

Python may be implemented by anyone on any platform. Consequently, and due to its significant popularity, you will find Python available on almost *every* computing platform. In addition to CPython, which is available for all major operating systems, including Windows, MacOS, and Linux, an implementation of Python called *IronPython* has been implemented for the .Net runtime [3], *Jython* has been implemented for the Java Virtual Machine (JVM) [4], and at least two Python subsets that run on microcontrollers: *MicroPython* [5] and *CircuitPython* [6]. A version of Python has even been implemented in Python itself, appropriately called *PyPy* [7].

More recently, CPython and many of its most important packages have been compiled to *WebAssembly* [8]. WebAssembly is a stack-based virtual machine that runs entirely in and is confined by, a web browser. The WebAssembly port of Python is called *Pyodide* [9]. Pyodide allows us to run Python in a web browser without the need to install it first. Pyodide is an important option for laboratory scientists because laboratory computers are often locked down for security reasons. Typically, the primary purpose of a lab computer is to operate an attached instrument or process data, not to perform general-purpose computing. For security reasons, it is often the case that installing new software is strictly forbidden. Fortunately, because the entire Pyodide Python environment may be loaded into a web browser directly, Pyodide provides a means to access the power of Python without the need to convince your IT team to grant you the elevated privileges required to install software.

No matter which implementation of Python you choose, your knowledge of the Python programming language will be instrumental in helping you streamline your laboratory operations.

## 1.2 Installing the Python Toolkit

To install CPython, visit the official Python home page at <https://www.python.org/> and click the Downloads link [2]. An appropriate installer for your operating system will be offered. Download and run the installer. Python will be installed for you on your computer.

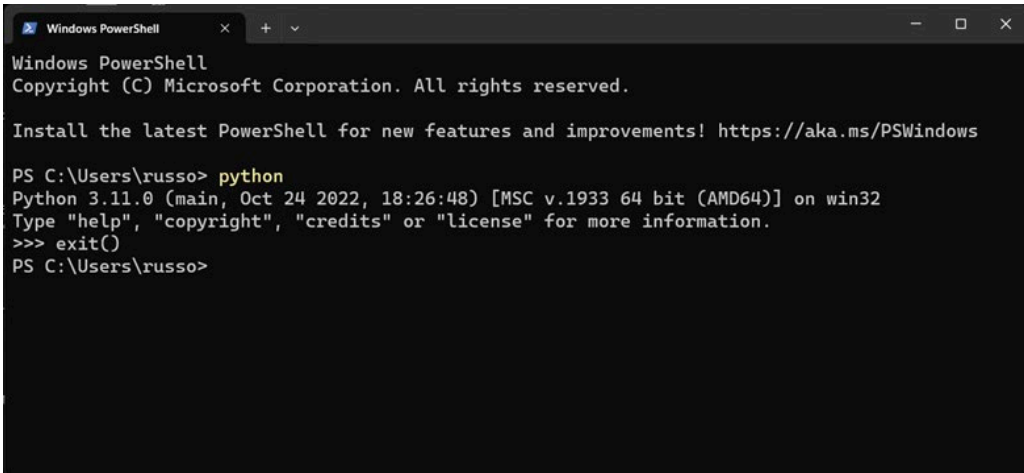
To test your installation, open a terminal program and enter the `python` command. Many terminal programs are available and will change based upon your computer's operating system. On MacOS and Linux, you should find *Terminal* as one of your program options. On Windows, you may use *PowerShell*, *Command Prompt*, or another option. But no matter which terminal program you use, simply enter the `python` or `python3` command into the terminal program and press *Enter*. This command will run the *Python Interactive Console*, which you may use to execute Python commands interactively.

To exit the Python console, enter the Python command `exit()`. See Figure 1.1 for an example.

If the installation of Python is successful but you are still having problems starting the Python console, our experience suggests that the problem lies with your operating system's ability to find the Python executable. Investigate where Python was installed and make sure that the path location is included in your PATH environment variable. Also check that you have the necessary permissions to access and execute the `python` program.

## 1.3 Python 3 vs. Python 2

Python 3 was introduced to the community in 2008 as a “breaking change” version of Python. Programs written for the previous Python 2 would not run in Python 3 due to significant changes to syntax and other implementation details. This change was necessary because several of the design decisions made for Python 2 needed an upgrade to make the language more suitable for modern



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\russo> python
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
PS C:\Users\russo>

```

**Figure 1.1** The Python Interactive Console running in Windows PowerShell.

applications. Some changes made to Python 3 were fundamental, including the way binary data is stored and processed.

There was a significant number of existing Python 2 programs in production around the world when Python 3 was announced. It is no surprise that many Python 2 programmers were less than enthusiastic about porting their source code to Python 3. Nevertheless, Python 2 was scheduled to be retired in 2015, but the resistance was so strong that this date had to be delayed. It wasn't until January 1, 2020 that Python 2 was finally and fully retired.

Even though Python 2 has been retired and it no longer receives security patches, you can still install and use it. If you have a version of Python 2 installed, please resist the urge to use it to write new programs and install Python 3 instead. For guidance porting Python 2 code to Python 3, refer to Python's own porting guide [10]. If you need both versions of Python available, you may install Python 3 with Python 2 and use them both simultaneously. With both versions of Python installed, run Python 3 from a terminal using the `python3` command in place of the `python` command. To make sure you have a recent version of Python 3 installed, you can enter the following command into a terminal. In this book we use Python 3 exclusively.

```
python --version
```

## 1.4 Python Package Index

One of Python's mottos is "batteries included," and for a good reason. A Python distribution comes with a huge library of prewritten modules for you to use and build upon. While it's true that a Python distribution includes quite a few "batteries," it is not possible to include them all.

If a module is not shipped with Python, there is a good chance someone in the Python community has contributed a module that will help you. Additional Python modules are distributed through a Python package repository, with the two most popular being the PyPI [11] and the Package Repository for Anaconda [12]. Anaconda is an exceptional platform that provides high quality Python installations, package distributions, and other open-source resources. Importantly, it also offers paid support

plans, which may be critical for businesses that depend upon Python as part of their core operations. In the following, we describe how to use PyPI for installing additional Python packages.

The PyPI provides a way for package authors to post their open-source Python packages, and for package users to find and install Python packages that are not distributed with Python. The PyPI hosts over 600,000 freely available Python packages ready for you to install and use. If you need something specific, there is a good chance that the PyPI has a package for that.

Python packages are downloaded from the PyPI and installed in your computing environment using a program and module distributed with Python called `pip` (*package installer for Python*). If you have a Python distribution, you have `pip`.

As an example, consider the `watchfiles` Python package, which is a toolkit for monitoring and responding to file system events [13]. The `watchfiles` package can help if you may want to execute a Python program automatically to parse data when a new file is written. Before you can use `watchfiles`, you must install it with the `pip` module. Python ships with a script named `pip` that is a convenient way to invoke the `pip` module. Our experience has been that this can be troublesome in some environments. To avoid the trouble, we skip the script and invoke the module directly to perform an installation from the PyPI using a command like the following.

```
python -m pip install watchfiles
```

This installs the `watchfiles` package so that it can be used by anyone logged in to the computer. To limit the installation to yourself only, add the `--user` option to the command.

```
python -m pip install --user watchfiles
```

To install a particular version of a package, add the package version to the `pip` command. For example, if you want to install version 0.24 of `watchfiles`, execute the following terminal command, which ensures that the version is exactly equal to 0.24.

```
python -m pip install --upgrade watchfiles==0.24
```

The `pip` package is updated frequently. When executed, it checks if there is a more recent version available and lets you know if there is. To ensure that you have the latest version of `pip` installed (or any module), add the `--upgrade` option.

```
python -m pip install --upgrade pip
```

If you are working in a networked environment that is behind a proxy server, which is the case with most sizable organizations, the `pip` command may be unable to access the PyPI repository directly. Fortunately, if you provide proxy server parameters to `pip`, it will use the proxy server to access PyPI. Add the `--proxy` option to the `pip` command along with the *proxy server domain name* and *port number*, as well as *username* and *password*, if necessary, using the following syntax.

```
python -m pip install packageName --proxy [user:passwd@]proxy.server:port
```

For many more details describing the options for installing Python packages from the PyPI, refer to the *Python Packaging User Guide* [14].

## 1.5 Programming Editors

In the next chapter you will learn that Python depends heavily on indentation to properly parse and interpret its source code structure. Statements in the same code block must be indented uniformly, otherwise they are considered to be in different code blocks, which often results in an incorrect interpretation of what you intended your program to do. Leading whitespace like tabs and spaces can be extremely important to Python programs. You must get it right.

It is possible to write Python programs using a simple text editor, but it can be difficult to ensure that all indentations are correct. An extra leading space character on one line can completely change the meaning of your program or break it entirely. For example, a plain text editor might use a sequence of space characters or a tab character to indent a statement on a line. Two consecutive lines of code, one above the other but using different whitespace characters for indentation, can appear to be indented identically. Despite the way the lines look, Python will consider these two lines to be in different code blocks. To ensure you don't end up with an incorrect Python program due to line indentation inconsistencies, it is strongly recommended that you write your programs using an editor designed for programming in Python. Fortunately, there are many good options.

One good option that is freely available is the *Visual Studio Code* editor, often called *VSCoDe* [15]. Note that VSCoDe is *not* the same as the *Visual Studio Integrated Development Environment*, which is far more sophisticated than VSCoDe. Because VSCoDe is a general-purpose programming editor, after installation it is necessary to add extensions to the editor specifically for Python. A good extension for Python programming is the *Python extension for Visual Studio Code* distributed by Microsoft, which is freely available through the *Visual Studio Code Marketplace* [16]. This extension adds several functions to VSCoDe for Python programming, including syntax highlighting and checking, debugging, outlining and navigation, formatting, refactoring, variable explorer, and more.

To install VSCoDe, visit <https://code.visualstudio.com/> and download an installer for your operating system. Run the installer to set up the editor on your computer. After installation is complete, start the program and find the icon bar docked to the left side of the window. Click the icon for Extensions, or enter Control-Shift-X. The Extensions icon is composed of four small boxes, one of which is being added to the others. Clicking this icon opens the Extensions Marketplace panel. Search for “Python” and find the “Python extension for Visual Studio Code” extension. Click the [Install] button. Once installed, this extension will configure your editor for advanced Python program editing, syntax highlighting, and much more. It is well worth the effort to seek out and install the VSCoDe extension pack for Python.

## 1.6 Notebook Editors

When working with experimental data, it helps to use a different style of editor called a Notebook. The idea was first described by Donald Knuth [17] when he introduced the concept of *literate programming*. In that paper, Knuth proposed extending our attitude toward computer programming as merely a way to instruct a computer but also to include a description for humans that explains a larger concept that is supported by the program. In this way, we consider such an extended computer program a work of literature.

The modern version of literate programming is best embodied in the Notebook-style editor, which intermingles formatted text with code snippets that analyze data and generate illustrative output. For Python, Jupyter and JupyterLab are among the most popular Notebook editors [18]. JupyterLab itself is composed of a Jupyter notebook coupled with a file system browser and other utilities. Jupyter is the *de facto* standard Notebook editor for Data Scientists who use Python.

Jupyter and JupyterLab leverage a web browser for its user interface, with a computer language *kernel* running behind it. A *kernel* is the compute engine that executes code snippets. Once the user enters code into a *code cell* of a notebook and submits it, the code is transmitted by the notebook to its running backend kernel. The kernel executes submitted code and results are returned to the notebook. If appropriate, returned results are rendered in a new notebook cell. For example, the result of a computation might be a chart or interactive widget. This architecture is illustrated in Figure 1.2.

To install Jupyter and JupyterLab, open a terminal and use the `pip` module to install the package.

```
python -m pip install jupyterlab
```

Once installed, Jupyter may be started by executing the following command in a terminal.

```
jupyter-notebook
```

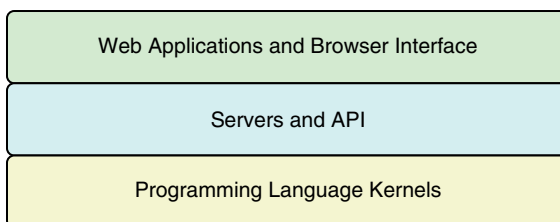
To start JupyterLab, replace “notebook” with “lab,” as follows.

```
jupyter-lab
```

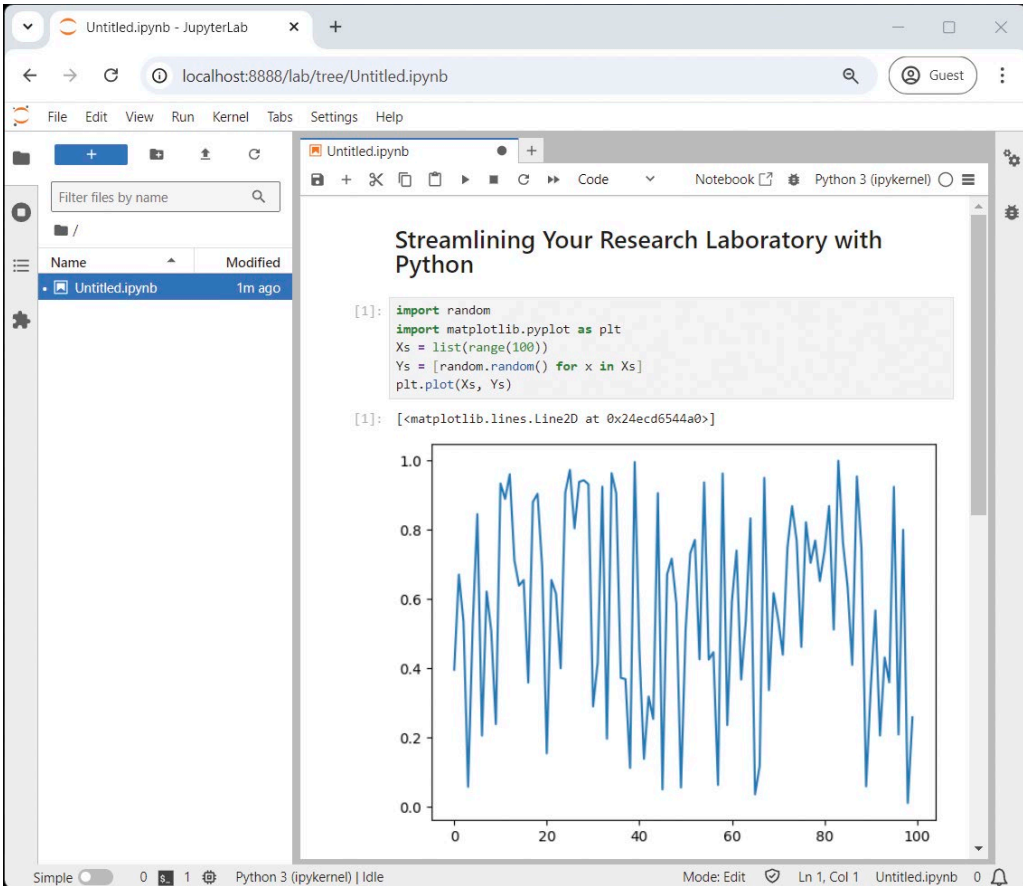
In both cases, a local notebook server program with the Python language kernel is started, followed by the opening of a browser window, which is automatically navigated back to the notebook server to load the initial interface.

To shut down the notebook server, click the [Quit] button in a Jupyter Notebook, choose the “File | Shut Down” menu option from JupyterLab, or simply enter Ctrl-C in or Cmd-C the terminal window that is running the notebook server program.

The modular architecture of Jupyter allows one kernel of a notebook to be swapped out for another kernel. With a standard communication protocol in place for transmitting code snippets to a kernel and returning results, it is possible to develop kernels for other programming languages in addition to Python. Currently, notebook kernels are available for Perl, Ruby, JavaScript, Fortran, Java, and other languages. The landscape of available kernels is advancing all the time. If you have an interest in a kernel for another language, consult the Jupyter Project documentation [18].



**Figure 1.2** Jupyter notebook architecture.



**Figure 1.3** JupyterLab in action.

Figure 1.3 demonstrates a simple example of JupyterLab with formatted descriptive text, a short Python program, and a chart resulting from the execution of the code in the code cell. This notebook illustrates how we may intermingle text, programs, and dynamically generated results, in a single document, thereby embodying Knuth’s concept of *literate programming*.

## 1.7 Using the Jupyter Notebook Interface

Notebook documents are composed of a linear stack of three kinds of cells: *Markdown cells* (formatted text), *code cells* (program snippets), and *raw cells* (unprocessed text). These cells may be added to a notebook, deleted, edited, and rearranged at will. Individual cells in a notebook may be executed in any order, or all cells may be executed from the top of the notebook to the bottom.

The content of any cell in a notebook is entered directly through the notebook interface. Executing a cell, whether it is a Markdown cell or a code cell, is performed by entering the key sequence Shift-Enter. This will either process and display the formatted text of a Markdown cell

as HTML or it will transmit code to the language kernel for execution with results displayed upon return. To learn more about Markdown text formatting syntax implemented in a Jupyter Notebook, explore the Jupyter Notebook documentation site [19]. Python language syntax will be explored in great detail in the chapters that follow.

From the perspective of a laboratory scientist, it is easy to see how a notebook may be used to perform the kind of analysis and results that are entered into a classic laboratory notebook. A Jupyter notebook used to document an experiment might open with formatted text describing the experiment performed, including detailed procedures, source materials, and an experimental design. Experimental data might be added to a cell in a tabular format suitable for processing. A code cell should contain the full source code for a short program used to process experimental data. Capturing the actual source code for the program used to process raw experimental data has the side benefit of documenting the analysis in a way that allows it to be reproduced easily. After analysis, additional code cells might generate charts, tables, and other illustrations rendered right in the notebook. Once the notebook document is written and tested, it might be possible to reuse it by updating experimental data and rerunning the entire notebook.

Of course, notebook documents like those created with JupyterLab are not and cannot be used as laboratory notebooks because they lack the required tracking, signature capabilities, and other security features. But they can form the basis of a report to be entered into a notebook, complete with an ability to reproduce the analysis at a later time by rerunning a copy of the notebook. It's easy to see how notebook editors can be a valuable tool for a research scientist.

## 1.8 JupyterLite

Earlier we described one implementation of Python called Pyodide. The authors of Pyodide recognized that several of Python's core data processing libraries could reach even more users if compiled to a WebAssembly target so that they were available to run in Pyodide. These additional packages are the tools used most often by Data Scientists. With these additional packages compiled to WebAssembly, and because JupyterLab uses the browser as its user interface, it was possible to combine JupyterLab with Pyodide and a set of additional libraries compiled to WebAssembly to create a fully in-browser data science environment. Add a lightweight browser-based file system, and the result is an environment named *JupyterLite* (Figure 1.4) [20].

JupyterLite is a complete notebook environment that runs entirely in a web browser – no installation necessary. This includes both the notebook front end as well as the Python kernel language back end. You may load this complete environment at the Jupyter website [21]. This is a tremendous benefit, especially when you are blocked from installing software, such as in a tightly controlled lab computing environment managed by large organizations.

Note that the JupyterLite environment is indeed a “lite” version of the toolkit. For example, the file system is simulated and exists entirely in browser storage. It is possible to upload files from a local disk to the browser-based simulated file system and download files from the browser to local disk, but accessing the local file system from the JupyterLite environment directly is impossible due to the tight security constraints implemented by web browsers. This limits the amount of data that can be analyzed. Also, while any pure Python package may be used in JupyterLite, more high-performance packages that depend on utilities written in another language, such as C or Rust, are not available unless they have been precompiled to WebAssembly.

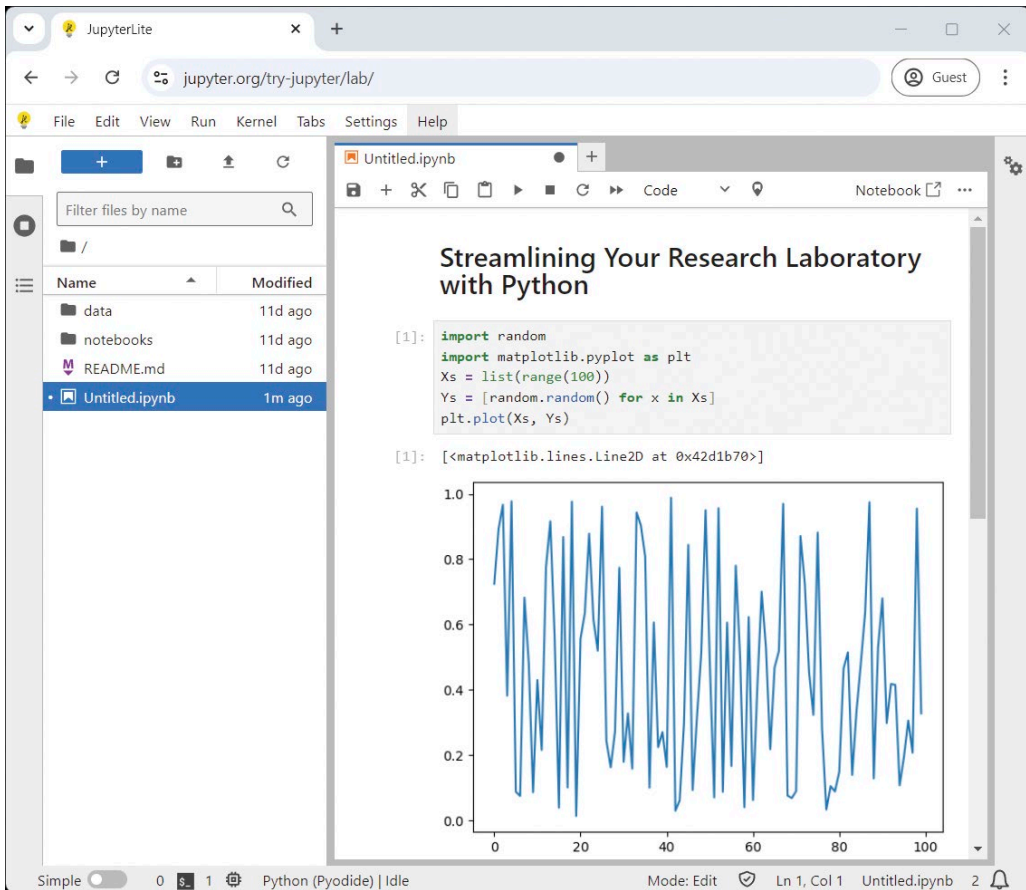


Figure 1.4 JupyterLite demonstration.

## 1.9 Things Change

Computing environments develop and change quickly. Unfortunately, this means that the information here may become out of date. It is worth taking some time to research the implementations of Python, programming editors, and other tools currently available. Better options may appear, and you will want to benefit from them.

### 1.10 Key Takeaways

1. Currently, Python is one of the most popular general-purpose programming languages.
2. Learning Python will provide you with the power to solve virtually any computing problem you may encounter.
3. The PyPI includes approximately 600,000 additional packages for you to download.
4. Python is one of the two most popular languages used for Data Science.

5. Python is an open-source language, which means it is free to use and has a large and active community of developers.
6. There are many implementations of Python for you to use. The CPython implementation is the reference standard by which other implementations are compared.
7. Install Python by visiting the language home page [2], clicking the Downloads link to download a suitable installer and running the installer program.
8. Additional packages are installable from the PyPI using the `pip` module, provided with the standard Python distribution.
9. Because Python syntax depends heavily on uniform indentation, a good programming editor will be a great benefit. Visual Studio Code is one very good option. To install VSCode, download an installer [15].
10. Notebook editors are another style of editor that support *literate programming*, a style of programming in which documentation, code, and results are intermingled.
11. The JupyterLab notebook editor is a very popular notebook style of editor used for Python programming in the field of Data Science. Notebooks are useful for research scientists as well.
12. Things change quickly in technology fields. Always check for the most recent releases and the latest options available for solving computing-related problems.

## 2

### Language Basics

In this chapter we introduce the basics of Python programming with variables and expressions. Fundamental Python data types are described along with how they may be combined into expressions using operators and global functions.

#### 2.1 Python Interactive Console

Even if you never write a complete Python program, being able to write and evaluate Python expressions interactively will give you an immediate advantage. You can use the Python Interactive Console to perform a wide range of ad hoc calculations, just like you would with an interactive calculator. If you need to calculate a dilution volume or the number of tubes or microplates required to perform an experiment, you'll find yourself popping open the Python Interactive Console to perform a quick calculation.

To start the Python Interactive Console, open a terminal program and enter the command `python` (or `python3`). If you are using Microsoft Windows, you can run PowerShell or Command Prompt as your terminal. On a Macintosh or Linux computer, the standard terminal program will do the trick. After entering the `python` command, the Python console will start in your terminal and offer you the `>>>` prompt. This is where you can enter any Python statement you want to be evaluated, even multi-line statements.

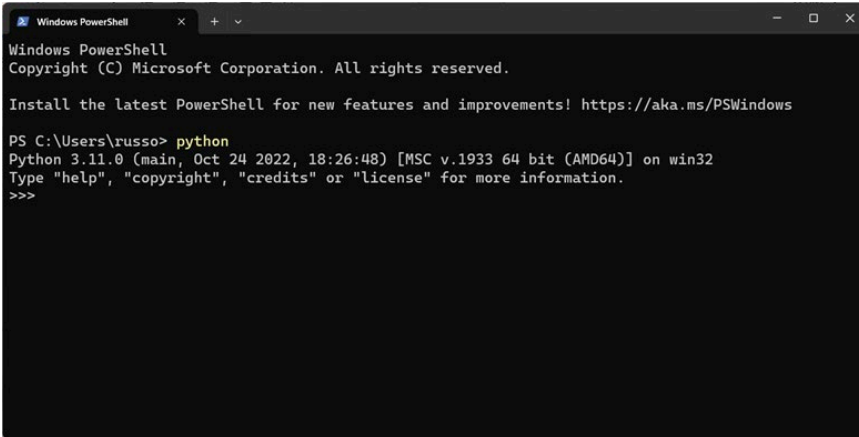
Figure 2.1 demonstrates the Python console started in Windows PowerShell. It will look the same, no matter which terminal program or operating system you use. To exit the console, enter the `quit()` or the `exit()` command and press the Enter or Return key. Make sure to add a pair of parentheses after the command and before pressing Enter or Return.

With the Python console running, go ahead and enter an expression. You will find that performing basic calculations is straightforward.

```
>>> 1.1 * 65000 / (5 * 8 * 52)
34.375
```

#### 2.2 Data Types

Python provides several different fundamental types of data for you to work with, right out of the box. These include three kinds of numbers, including integers, floating-point numbers, and even complex numbers. Other built-in data types are strings of characters, Boolean values



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\russo> python
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>

```

**Figure 2.1** Python Interactive Console.

(True and False), and a special keyword used to indicate the *absence* of a value, called None. These are summarized in Table 2.1.

The first column of Table 2.1 lists the *class* of the Python data type. These predefined Python terms will be useful to you. The second column provides a description of each data type, the third column indicates the approximate range of values that can be represented by the type, and the last column provides several examples of each data type written in a way that is recognized by Python.

Every data type is stored in a finite amount of memory, which is why the range of values that can be represented by each type is limited. We won't discuss how these fundamental data values are stored, but it is important to understand that every value has a data type, and each type has its own way of encoding itself in memory. It is worth verifying that the types chosen to represent values used in your calculations will not approach the limits given in the third column of Table 2.1, either from the perspective of a range or in the number of required significant digits.

**Table 2.1** Fundamental Python data types.

Data type	Description	Approximate range	Literal examples
int	An integer. A whole number with no decimal or fractional part.	-9223372036854775808 to 9223372036854775807	0, 1, -2, 1234
float	A base-10 number with a decimal and optionally places to the right of the decimal.	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	0.0, 0.1, 3.1415926, 10., -2e3
complex	A complex number with both real and imaginary parts.	Same as float, for real and imaginary parts,	1j, -1 + 0j, -1.2 + 3.4j
str	A sequence of 0 or more characters delimited by pairs of matching single quotes or double quotes.	Any sequence of characters from a character set.	"hello", 'goodbye', "", "A", '@', "12"
bool	One of the Python values True or False.	True or False	True, False
NoneType	The Python value None.	None	None