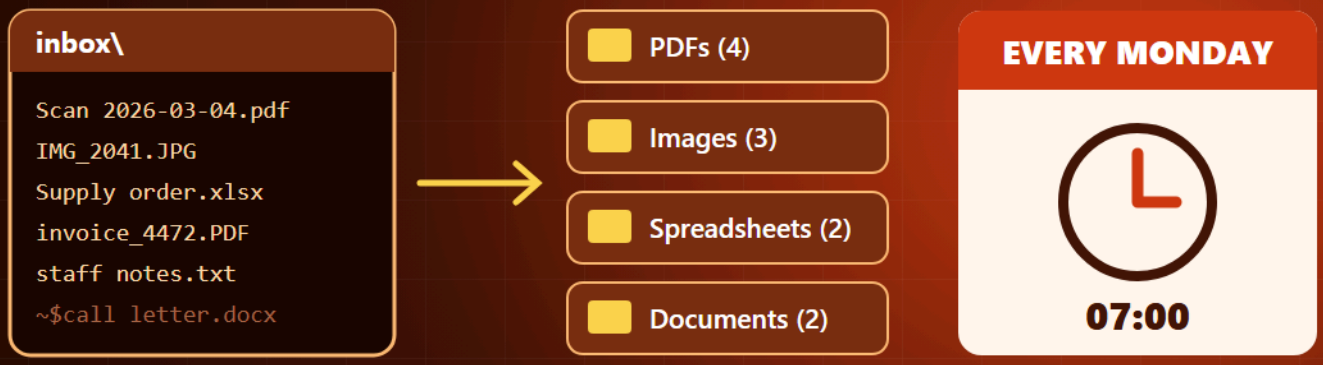



```
for pdf in inbox.glob("*.pdf"):
    inv = parse_invoice(pdf)
    total += inv.amount
wb.save("weekly.xlsx")
smtp.send_message(msg)
# dry run first, then --apply
```

Clinic	Production
Riverside	$5,835.00
Northgate	$7,610.00
Lakeview	$3,800.00
Total	**$17,245.00**

Weekly production report - week of March 9, 2026
weekly_production.xlsx · weekly_memo.docx · sent at 07:00

A PRACTICAL, HANDS-ON GUIDE

Python Automation for Everyday Work

Automate the repetitive work in your office — safely, step by step

Files · Folders · CSV · Excel · Word · PDF · Images
Email · Web APIs · argparse · Logging · Scheduling

FILES · SPREADSHEETS · DOCUMENTS · EMAIL · SCHEDULES

THENEXBOOKS

Python Automation for Everyday Work

Automate Files, Spreadsheets, PDFs, Word Documents, Email, and Scheduled Tasks
From a messy shared folder to a tested, scheduled toolkit the whole office can run

First Edition

THENEXBOOKS

Python Automation for Everyday Work
Automate Files, Spreadsheets, PDFs, Word Documents, Email, and Scheduled Tasks

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Python, its standard library, and third-party packages change over time; behavior, defaults, and available features can differ between versions. Always test automation on copies of your files, keep backups, and check the documentation for the versions you use. Nothing in this book is legal advice; follow your organization's policies and the privacy and email laws that apply to you.

Python is a registered trademark of the Python Software Foundation. Microsoft, Windows, Excel, Word, Outlook, OneDrive, SharePoint, and Microsoft 365 are trademarks of the Microsoft group of companies. macOS is a trademark of Apple Inc. Linux is a registered trademark of Linus Torvalds. Other product and project names, including openpyxl, python-docx, pypdf, Pillow, requests, pandas, matplotlib, and pytest, belong to their respective owners. Their use in this book is for identification only and does not imply endorsement. This book is independent and is not affiliated with, sponsored by, or endorsed by any of these organizations or projects.

Calloway Dental Group, the dental practice used as the running example in this book, is fictional, as are its clinics, staff, patients, suppliers, appointments, invoices, and all sample data. Procedure codes are invented for the book and are not real billing codes. Email addresses use reserved example domains and phone numbers use the fictional 555 range.

Preface

Every office has a list of chores that nobody wrote down but everybody does. Somebody downloads the same three exports every Monday and pastes them into a spreadsheet. Somebody renames scanned invoices one by one so the bookkeeper can find them. Somebody copies names and dates from a list into a letter template, prints, checks, and fixes the one where the date came out wrong. None of these jobs is hard. All of them are slow, repetitive, and easy to get slightly wrong on a busy day.

If you know a little Python, you already have most of what you need to make those chores disappear. Python's standard library handles files, folders, CSV, zip archives, email, and scheduling, and a handful of well-known packages handle Excel workbooks, Word documents, PDFs, images, and web APIs. What is usually missing is not syntax but practice: how to plan an automation, how to make it safe to run on real files, how to make it produce output people trust, and how to hand it to colleagues who will never open a code editor.

This book fills that gap. We automate one realistic office from start to finish, one chore at a time, and every chapter ends with something that works: a script that sorts a shared folder, a report workbook with formatting and a chart, letters generated from a Word template, invoices read straight out of PDFs, reminder emails tested safely on your own computer, and a scheduled job that has the Monday report waiting before anyone arrives.

Who This Book Is For

- **Office workers and administrators** who know basic Python, or have finished a beginner course, and want to use it on the files they handle every day.
- **Analysts and coordinators** who live in spreadsheets, exports, and email attachments and want reports that build themselves.
- **Small-business owners and managers** who want to remove repetitive work without buying another subscription.
- **IT generalists and "the person who is good with computers"** who are asked to automate things for colleagues and need the result to be safe, documented, and easy to run.

You should be comfortable with variables, lists, dictionaries, loops, functions, and importing modules. We do not re-teach the language; we show how to apply it.

What You Will Learn

- How to choose what to automate and how to make automations safe with copies, dry runs, backups, and tests.
- How to find, read, write, sort, rename, copy, and archive files with `pathlib`, `shutil`, and `zipfile`.

- How to read and combine CSV exports correctly, with real money and date handling, and how to read and build Excel workbooks with openpyxl, including styles, number formats, formulas, and charts.
- How to summarize office data quickly with pandas and how to clean messy text with regular expressions.
- How to read, edit, and generate Word documents with python-docx, including mail-merge letters from a template.
- How to merge, split, rotate, stamp, protect, and extract text from PDFs with pypdf, and how to resize and convert photos with Pillow.
- How to send and read email with `smtplib`, `email`, and `imaplib` — tested against a local server, never a real mailbox.
- How to call web APIs with requests, with timeouts, retries, and proper error handling.
- How to turn scripts into reliable tools: command-line options with argparse, configuration files, environment variables, logging, scheduling with Task Scheduler, cron, and the schedule library, packaging for coworkers, and a small tkinter launcher.

How This Book Is Organized

Part I — Files and Folders (Chapters 1–5) sets up your workbench and automates the shared folder: paths, reading and writing files, sorting, moving, renaming, zip archives, and backups. **Part II — Spreadsheets and Data** (Chapters 6–10) covers CSV exports, reading and building Excel workbooks, quick summaries with pandas, and text cleanup with regular expressions. **Part III — Documents, PDFs, and Images** (Chapters 11–15) covers Word documents and template letters, PDF processing and invoice extraction, and photo processing. **Part IV — Email and the Web** (Chapters 16–19) covers sending and reading email safely and calling web APIs. **Part V — Reliable Scripts** (Chapters 20–24) turns scripts into dependable tools with command-line options, configuration, logging, scheduling, and safety nets. **Part VI — Sharing and Case Studies** (Chapters 25–28) shows how to share your tools with coworkers, adds a simple graphical launcher, and walks through two complete case studies. The appendices provide a cheat sheet, templates and reference tables, a production checklist, and a glossary.

Each chapter includes exercises in a **Try It Yourself** section and ends with a **Key Takeaways** box for quick review.

The Running Example: Calloway Dental Group

Throughout the book we work for **Calloway Dental Group**, a fictional dental practice with three clinics — Riverside, Northgate, and Lakeview. Its operations manager, Dana Whitfield, spends too much of every week on chores: a shared inbox folder full of scans, photos, and spreadsheets with random names; fifteen appointment exports to combine every Monday; a supplies workbook to check for low stock; recall letters and reminder emails for patients who are due for a checkup; lab invoices

that arrive as PDFs; clinic photos that are too large for the website; and supplier prices to look up. Chapter by chapter, Dana replaces those chores with a small Python toolkit, `calloway-tools`, that the whole office can run. The sample data is small enough to check by hand, and every output shown in this book was produced by running the code.

Versions and Conventions

Examples target **Python 3.12** and also run on newer versions. Where a newer Python release adds something useful, a Note says so. Third-party packages — openpyxl, python-docx, pypdf, Pillow, requests, schedule, and pandas — are used through their stable, widely documented features; recent versions of each work. The book is written for **Windows**, the most common office setup, and notes the differences for macOS and Linux where commands change. On Windows we start Python with the `py` launcher; on macOS and Linux use `python3`.

All output in this book uses US English formats — dates such as `03/09/2026` or `March 9, 2026` and money such as `$1,234.50` — produced by small helper functions so the result is the same on any computer, whatever its regional settings.

> **TIP**
> A recommended practice or a technique that makes your work easier.

> **NOTE**
> Background information, version differences, or a useful aside.

> **WARNING**
> A common mistake or something that can cause data loss, wrong results, or security problems.

Code appears in `monospace` blocks with a caption naming the file, such as `calloway/appointments.py`. Commands you type are captioned **Terminal**, and what the program prints appears in a separate block captioned **Output**. Long lines are wrapped to fit the page.

Contents

Preface

PART I — FILES AND FOLDERS

1 Automating Everyday Work: What to Automate and Meet Calloway Dental Group

2 Your Automation Workbench: Python, the py Launcher, Virtual Environments, and the Sample Project

3 Paths and Files with pathlib: Finding, Reading, and Writing Files Safely

4 Organizing Folders: Sorting, Moving, and Copying Files with shutil

5 Bulk Renaming, Zip Archives, and Backups

PART II — SPREADSHEETS AND DATA

6 CSV Files Done Right: Reading, Cleaning, and Combining Daily Exports

7 Reading Excel Workbooks with openpyxl

8 Building Excel Reports: Styles, Number Formats, Formulas, and Charts

9 Quick Summaries with pandas: Pivot Tables for Office Data

10 Text Cleanup and Regular Expressions: Names, Phone Numbers, Dates, and Codes

PART III — DOCUMENTS, PDFS, AND IMAGES

11 Word Documents with python-docx: Reading, Editing, and Building Reports

12 Mail-Merge Letters: Filling Word Templates from Data

13 PDFs with pypdf: Merge, Split, Rotate, and Extract Pages

14 Reading Invoices from PDFs: Text Extraction, Renaming, Stamps, and Passwords

15 Images with Pillow: Resizing, Converting, Orientation, and Contact Sheets

PART IV — EMAIL AND THE WEB

16 Sending Email Safely: smtplib, EmailMessage, and a Local Test Server

17 Better Email: Attachments, HTML, Templates, and Bulk Reminders

18 Reading Email: The email Package, Saved Messages, and IMAP

19 Web APIs with requests: JSON, Parameters, Errors, Timeouts, and Retries

PART V — RELIABLE SCRIPTS

20 Command-Line Scripts with argparse: Options, Subcommands, and Exit Codes

21 Configuration Files, Environment Variables, and Secrets

22 Logging and Error Handling for Unattended Scripts

23 Scheduling: Windows Task Scheduler, cron, and the schedule Library

24 Safety Nets: Dry Runs, Backups, Idempotent Scripts, and Tests with pytest

PART VI — SHARING AND CASE STUDIES

25 Sharing Scripts with Coworkers: pyproject.toml, Entry Points, zipapp, and Launchers

26 A Friendly Front End: A Small tkinter Launcher for Your Scripts

27 Case Study: The Monday Morning Report Pipeline

28 Case Study: Month-End at Calloway Dental Group, and Where to Go Next

APPENDICES

A Python Automation Cheat Sheet

B Templates and Reference

C Automation Production Checklist

D Glossary

PART I

Files and Folders

Choosing what to automate, setting up a clean Python workbench, and taming the shared folder: finding and reading files safely, sorting and moving them with a dry run first, renaming in bulk, and protecting everything with zip archives and timestamped backups.

1 · Automating Everyday Work

2 · Your Automation Workbench

3 · Paths and Files with pathlib

4 · Organizing Folders

5 · Renaming, Zip Archives, and Backups

CHAPTER 1 · PART I — FILES AND FOLDERS

Automating Everyday Work: What to Automate and Meet Calloway Dental Group

Before we write a single line of automation, we need to know which jobs deserve a script, which do not, and how to change real office files without ever putting them at risk. This chapter introduces the dental practice whose office we will automate, maps every kind of file to the Python library that handles it, and ends with a first small script that looks inside a messy shared folder.

Why Automate the Boring Parts?

Every office has a layer of work that nobody chose and nobody enjoys. Someone downloads the same three reports every morning. Someone renames scanned invoices so they sort by date. Someone copies numbers from one spreadsheet into another, checks them twice, and still worries about a typo. Someone writes forty nearly identical letters where only the name, the date, and the clinic change. None of this is hard, but all of it takes attention, and attention is the one thing a busy office never has enough of.

You already know basic Python. You can write a loop, call a function, read a dictionary, and import a module. What you may not have seen yet is how directly those skills map onto this layer of office work. A folder full of files is just a list of paths. A spreadsheet is rows of values. A Word letter is a template with blanks. An email is a structured message with headers and a body. Once you see your work this way, a large share of the repetitive part becomes a short Python script that runs in seconds and makes the same decision the same way every time.

This book is about that translation. We will not build a web application or a data science model. We will automate files, spreadsheets, PDFs, Word documents, images, email, simple web APIs, and scheduled tasks, using tools that run on an ordinary office PC. Each chapter takes one real task from start to finish, shows the safe way to do it first, and shows the real output so you know what success looks like.

Automation is not only about speed. A script does not get tired on Friday afternoon, and a script that writes a log tells you exactly what happened while you were away. Consistency and traceability often matter more than the minutes saved.

What Makes a Task Worth Automating

Not every annoying task is a good automation candidate. Before writing code, we run each idea through four simple questions. A task that scores well on all four is almost always worth a script. A task that fails two or more usually is not, at least not yet.

1. Is it frequent?

A job you do every day or every week repays the effort of writing and testing a script many times over. A job you do once a year may be faster to do by hand, especially if the details change every time. Frequency also matters for reliability: a script that runs every Monday gets exercised constantly, so problems surface quickly. A yearly script tends to break silently because the world changed since last year and nobody noticed.

2. Is it rule-based?

If you can write the steps on an index card without the words "it depends", the task is rule-based. "Move every PDF from the inbox into the PDFs folder" is a rule. "Decide which supplier gives us the best deal this quarter" is a judgment call. Scripts are excellent at rules and poor at judgment. Many real tasks mix the two; in that case we automate the rule-based part and leave a clear, short list for a person to review.

3. Is it error-prone when done by hand?

Copying amounts between files, retyping patient IDs, and renaming dozens of files are classic sources of small mistakes. A tired person makes them; a script does not. When a mistake is expensive, such as a wrong invoice total or a reminder sent to the wrong patient, automation is worth it even for a task that is only moderately frequent.

4. Are the inputs digital and reasonably stable?

A script needs something it can read: a CSV export, a workbook, a folder of files, a PDF with real text, or a web API. If the input is a phone call, a handwritten note, or a scanned image, you first need a human (or an extra tool such as OCR) to turn it into data. Stability matters too. If the export format changes every month, the script will need constant repairs. If it has looked the same for two years, you can rely on it.

> **TIP**
>
> Keep a simple list called "things I do more than once a week". Add a line whenever you catch yourself doing the same clicks again. After a month, the best automation candidates are obvious: they are the lines with the most tally marks.

What Not to Automate

Knowing when to stop is part of the craft. Some tasks look tempting but cost more to automate than they save, or carry risks that a script makes worse rather than better.

- **Judgment and relationships.** Deciding how to respond to an unhappy patient, negotiating with a supplier, or choosing which bill to pay first should stay with people. A script can gather the facts for them, but it should not make the call.
- **Rare, one-off jobs.** If you will do something once, do it by hand carefully. Writing, testing, and documenting a script for a single use rarely pays off unless the job is huge.
- **Unstable inputs.** If a vendor changes its report layout every few weeks, a script that depends on that layout becomes a maintenance burden. Ask for a stable export first.
- **Things another tool already does well.** If your team lives inside Excel, Power Query or VBA may be a better fit for some workbook chores, because they stay inside the file everyone already opens. We automate from Python because it handles many file types at once, but it is not the only good answer.
- **Anything you cannot check.** If you have no way to tell whether the script did the right thing, you are not automating a task, you are hiding it. Every script in this book prints or saves something a person can review.

There is also a quieter risk: automating a bad process. A script makes a confusing process faster, not better, so first ask whether the steps still make sense.

Meet Calloway Dental Group

Throughout this book we work for one fictional organization, so every example connects to the same people and the same files. **Calloway Dental Group** is a three-clinic dental practice in the fictional town of **Maple Falls** in the United States. It was founded by **Dr. Helen Calloway**, who still practices at the Riverside clinic.

Code	Clinic	Phone	Front-desk lead
RIV	Riverside	(555) 555-0190	Marcus Lee
NGT	Northgate	(555) 555-0191	Tom Becker
LKV	Lakeview	(555) 555-0192	Carla Ruiz

Each clinic has a front-desk mailbox: `riverside@calloway-dental.example`, `northgate@calloway-dental.example`, and `lakeview@calloway-dental.example`. The three-letter codes appear everywhere in the data, from appointment exports to supply sheets, so it is worth remembering them now: RIV, NGT, LKV.

The people

Dana Whitfield is the operations manager and the person who automates. She is comfortable with Python basics, she is the one coworkers call when "the computer is doing something weird", and she is tired of spending her Monday mornings copying numbers. **Ruth Morgan** is the part-time bookkeeper who receives month-end files. Dr. Calloway receives the weekly report and makes the decisions it informs.

The practice has six providers, and each has a three-letter code used in the appointment data: Dr. Helen Calloway (HCA) and hygienist Jordan Ellis (JEL) at Riverside, Dr. Samuel Okoye (SOK) and hygienist Ana Silva (ASI) at Northgate, and Dr. Mei Tanaka (MTA) and hygienist Chris Navarro (CNO) at Lakeview.

The outside companies

Two fictional suppliers send Calloway files and data. **Pinecrest Dental Lab** makes crowns and night guards and sends its invoices as PDFs. **Bluewater Dental Supply** sells gloves, masks, bibs, and other consumables, and offers a small price and order API; in Chapter 19 a local test server stands in for it. All company domains in this book end in `.example`, all patient addresses use `@example.com`, and all phone numbers use the 555-01xx range, so nothing in the examples can reach a real person.

The data

Calloway's software and staff produce a steady stream of files, and those files are the raw material for every chapter:

- **Daily appointment exports**, one CSV file per clinic per day, named like `appointments_RIV_2026-03-09.csv`. Each row is one appointment with a procedure code, a fee, and a status of `Completed`, `No-show`, or `Cancelled`.
- **A patient list**, `patients.csv`, with 30 patients (P1001 to P1030), their clinic, their last visit, and how often they should come back for a checkup.
- **A supplies workbook**, `inventory_2026-03.xlsx`, with one sheet per clinic listing what is on hand and when to reorder.
- **A Word template** for recall letters, the "time for your checkup" letters patients receive.
- **Lab invoice PDFs** from Pinecrest, an office handbook PDF, a few clinic photos, and some saved emails.
- **A shared inbox folder** where colleagues drop whatever they need Dana to deal with, in every naming style imaginable.

The practice uses a set of internal procedure codes in its exports. They are simple abbreviations chosen for this book, not official billing codes:

Code	Procedure	Fee
EXM	Periodic exam	$65.00

CLN	Adult cleaning	$110.00
XRY	Bitewing X-rays	$85.00
FIL	Composite filling	$185.00
CRN	Crown	$1,150.00
EXT	Simple extraction	$225.00
WHT	Whitening	$350.00

In Chapter 2 you will create all of these files on your own PC with one script, so you can run every example in the book against exactly the same data we used.

Dana's Task Inventory

Dana's first step was not code. She spent two weeks writing down every repetitive task she did, how often it came up, and roughly how long it took. The minutes below are **her own rough estimates** for her office, not measurements and not a promise of what you will save. Your numbers will be different, and that is the point of making your own list.

Task	How often	Dana's estimate	Chapters
Sort the shared inbox folder	Daily	about 15 min	3, 4, 5
Monday production report	Weekly	about 90 min	6, 8, 9, 11, 27
Recall letters and reminder emails	Weekly	about 60 min	10, 12, 16, 17
Rename and total lab invoices	Weekly	about 30 min	13, 14
Check supplies and reorder	Weekly	about 45 min	7, 19
Resize clinic photos for the website	Monthly	about 30 min	15
Month-end package for Ruth	Monthly	about 2 hours	5, 28

Look at what the list has in common. Every task starts from files that already exist in digital form. Every task follows rules Dana could explain to a new hire in a few minutes. Every task is repeated often enough that small mistakes add up. And every task produces something a person can check: a folder, a workbook, a letter, an email in an outbox. That is the profile of good automation work.

> **NOTE**
>
> Estimates like Dana's are useful for choosing what to do first, not for impressing anyone. A task you estimate at 90 minutes a week is a better first project than one at 10 minutes a month, even if the second one is more interesting to code.

The Toolbox Map

Python's standard library already handles files, folders, CSV, zip archives, email, and dates. For office formats such as Excel and Word, a handful of well-known third-party packages fill the gaps. Here is the map we follow throughout the book. You do not need to memorize it; it is here so you know where each chapter fits.

You have	We use	Chapters
Files and folders	`pathlib`, `shutil`, `os`	3, 4, 5
Zip archives and backups	`zipfile`, `shutil`	5
CSV exports	`csv`, `decimal`, `datetime`	6
Excel workbooks (.xlsx)	`openpyxl`	7, 8
Quick summaries and pivots	`pandas`	9
Messy text	string methods, `re`	10
Word documents (.docx)	`python-docx`	11, 12
PDF files	`pypdf`, `matplotlib`	13, 14
Photos and images	`Pillow`	15
Sending email	`smtplib`, `email`	16, 17
Reading email	`email`, `mailbox`, `imaplib`	18
Web APIs and JSON	`requests`	19
Commands and settings	`argparse`, `tomllib`	20, 21
Logs and schedules	`logging`, Task Scheduler, cron	22, 23
Tests and sharing	`pytest`, `pyproject.toml`	24, 25
A simple window	`tkinter`	26

Two things about this map are worth saying early. First, most of the tools are boring on purpose. They have been around for years, they are documented, and many office automation scripts on the internet use them, so when you search for help you will find answers. Second, the map is about file types, not about programming techniques. When a coworker hands you a new chore, the first question is "what kind of file is this?", and the map tells you where to start.

The Safety Mindset

A script that moves a hundred files moves them in a second. That is wonderful when the script is right and painful when it is wrong. Office files are often the only copy of something important, and patients' information deserves extra care. So before any technique, we adopt a few habits that every chapter

follows.

Work on copies first

While you develop a script, point it at a copy of the real folder. The sample data you create in Chapter 2 exists precisely for this. When the script behaves correctly on the copy, you point it at the real folder, and even then you start with a dry run.

Dry run by default

Every script in this book that moves, renames, or deletes anything first prints what it *would* do and changes nothing. Only when you add an explicit `--apply` option does it act. This sounds like a small thing, but it changes how you work: you read the plan, you spot the surprise ("why is the lock file in the list?"), you fix the rule, and only then do you let the script touch real files.

Back up before bulk changes

Before a script changes many files at once, it makes a backup. In Chapter 5 we build `make_backup()`, which zips a folder into a timestamped archive and checks that the archive is readable. A backup you have never tested is a hope, not a backup.

Never on the only copy

If a file exists in exactly one place, a script should read it, never modify it in place. Our report builders always write new files with dates in their names, such as `weekly_production_2026-03-09.xlsx`, and never overwrite the source workbook.

Least data, especially for patients

Calloway handles patient information, and healthcare privacy laws (HIPAA in the United States, and similar rules elsewhere) may apply to your office. This book does not give legal advice; ask whoever is responsible for compliance in your organization. What we can do as script writers is simple: export only the columns a script needs, keep patient details out of log files and email subject lines (a first name at most), store secrets such as passwords outside the code, and send every test email to a local test server instead of a real mailbox.

> **WARNING**
>
> Python's file functions do not use the Recycle Bin. When a script deletes a file with `Path.unlink()` or a folder with `shutil.rmtree()`, it is gone; there is no Undo. This is the main reason every destructive action in this book is a dry run unless you explicitly ask for it, and why backups come before bulk changes.

How the Book's Project Grows

We do not write throwaway snippets. Chapter by chapter, we build a small toolkit called **calloway-tools** that the whole office can eventually use. It lives in one project folder. In this book we use `C:\CallowayTools` as that folder; you may put it elsewhere, as long as you adjust the paths in the examples.

Project layout (finished state)

```
C:\CallowayTools\
  calloway\            the package: one module per job
    __init__.py        version number
    organize.py        sort the inbox (Chapters 3-4)
    renaming.py        bulk renames (Chapter 5)
    backup.py          zip backups (Chapter 5)
    appointments.py    CSV exports (Chapter 6)
    ...                one module per later chapter
    cli.py             the "calloway" command (Chapter 20)
  scripts\             your small chapter scripts
  tests\               automated tests (Chapter 24)
  tools\               make_sample_data.py (Chapter 2)
  data\                sample input files (created in Chapter 2)
  output\              files our scripts create
  backups\             zip backups (Chapter 5)
  logs\                log files (Chapter 22)
```

The work happens in three stages. In the early chapters you write small scripts in the `scripts\` folder and, little by little, move the reusable parts into modules in the `calloway\` package. In Chapters 20 to 22 those modules gain a proper command-line interface, a configuration file, and logging, so a single command such as `py -m calloway weekly-report` replaces a handful of scripts. In the final chapters we schedule the commands, test them, share them with coworkers, and put a small window in front of them for people who prefer buttons.

This order is deliberate. You will see the simple version of every idea before the polished one, and you will understand why each piece of structure exists, because you will have felt the problem it solves. When an early chapter shows a first version of a function, the text says so, and a later chapter improves it until it matches the finished project.

Part	What the toolkit gains
I. Files and Folders	Sorting, renaming, zip backups
II. Spreadsheets and Data	CSV, Excel reading and reports, pandas, text cleanup
III. Documents, PDFs, and Images	Word memos and letters, PDF invoices, web photos
IV. Email and the Web	Sending and reading mail, a supplier API
V. Reliable Scripts	Commands, configuration, logs, schedules, tests
VI. Sharing and Case Studies	Packaging, a launcher window, two full pipelines

Conventions Used in This Book

A few conventions keep the examples consistent. It is worth reading them once so nothing surprises you later.

Code, commands, and output

Every code listing has a caption above it. A caption such as `scripts/count_inbox.py` names the file, relative to the project folder. `Python (interactive)` marks lines typed at the Python prompt, shown with `>>>`. `Terminal (PowerShell)` marks commands typed in a PowerShell window, and `Terminal (macOS/Linux)` marks the equivalent for those systems where it differs. A block captioned `Output` shows what the program printed. We ran every example and pasted the real output; when a line was too long for the page we wrapped it and say so. On your PC some details will naturally differ, such as file dates or folder names, and the text points those out.

Windows first, with notes for macOS and Linux

Most office PCs run Windows, so terminal examples use PowerShell and the Windows Python launcher, `py`. The prompt looks like this:

Terminal (PowerShell)

```
PS C:\CallowayTools> py -m scripts.count_inbox
```

On macOS and Linux, type `python3` instead of `py` and use forward slashes in paths. Where commands differ in a more interesting way, such as activating a virtual environment or scheduling a job, the chapter shows both.

Dates and money in US format

Calloway is a US practice, so the book shows dates like `03/09/2026` and `March 9, 2026`, and money like `$1,234.50`. Your PC may use different regional settings, and Python can pick those up in surprising ways. To keep output identical on every machine, the project uses its own small helpers, `us_date()`, `long_date()`, and `format_usd()`, which we build in Chapter 6. File names always use ISO dates such as `2026-03-09`, because they sort correctly in any folder view.

Dry run and apply

When we say a command runs "as a dry run", it plans and prints but changes nothing. When we say "with `--apply`", it really does the work. You will see the phrase `(dry run; add --apply)` at the end of many outputs.

Python version

Everything in this book runs on Python 3.12. Recent versions (3.13 and later) add a few conveniences, which we mention briefly where they matter, but the code never depends on them. If your office installs a newer Python, the examples still work.

Your First Look Inside the Shared Inbox

Let us end with a small but real piece of automation. Calloway's front desks share a folder, `data\inbox`, where people drop files for Dana: scans, photos, exports, a letter template, whatever needs attention. Before we sort that folder (Chapters 3 and 4), we want a quick picture of what is in it. How many PDFs? How many images? Anything odd?

You can run this script once Chapter 2 has created the sample data in your project folder. If you are reading ahead, simply study it now and run it later.

scripts/count_inbox.py

```
"""Count the files in the shared inbox by extension (Chapter 1)."""

from collections import Counter
from pathlib import Path

INBOX = Path("data") / "inbox"

counts = Counter()
for path in INBOX.iterdir():
    if path.is_file():
        counts[path.suffix.lower() or "(none)"] += 1

for suffix, count in counts.most_common():
    print(f"{suffix:8} {count}")
print(f"{sum(counts.values())} files in {INBOX}")
```

Here is what each part does. `Path("data") / "inbox"` builds a path to the inbox folder relative to the folder you run the script from; the `/` operator joins path parts, and Chapter 3 explains it fully. `INBOX.iterdir()` yields one `Path` for every entry in the folder. We skip anything that is not a regular file, such as a subfolder. For each file, `path.suffix` gives the extension including the dot, such as `.PDF` or `.jpg`, and `.lower()` makes `.PDF` and `.pdf` count as the same thing. A file with no extension would have an empty suffix, so we label it `(none)`.

The counting is done by `collections.Counter`, a dictionary subclass designed for exactly this job. Reading a missing key from a `Counter` gives zero instead of raising `KeyError`, so `counts[key] += 1` works the first time a key appears. `most_common()` returns the pairs from the highest count to the lowest. The format specification `{suffix:8}` pads the extension to eight characters so the numbers line up in a column.

Run it from the project folder. Notice the form of the command: we start every chapter script as a module, with `-m` and a dotted name (`scripts.count_inbox`, no `.py`), rather than by its file path. Chapter 2 explains why this matters as soon as a script imports our own package:

Terminal (PowerShell)

```
PS C:\CallowayTools> py -m scripts.count_inbox
```

Output

```
.pdf     4
.jpg     2
.docx    2
.csv     1
.ini     1
.png     1
.zip     1
.txt     1
.xlsx    1
14 files in data\inbox
```

Fourteen files, and already some useful discoveries. There are four PDFs, but if you look in the folder, one of them is named `invoice_pinecrest_4472.PDF` in capitals; without `.lower()` it would have been counted separately. There are two `.jpg` files, and again one of them is really `IMG_2041.JPG`. There is a `.ini` file, which turns out to be `desktop.ini`, a hidden file Windows creates to remember folder settings. And one of the two `.docx` files is `~$call letter template.docx`, a tiny lock file that Word leaves behind while a document is open (or after it crashes). Neither of those last two is something Dana should sort into a folder.

This is exactly why we look before we act. A sorting script written without this glance might have moved the Windows settings file and the Word lock file into "Other" or "Documents", confusing Windows and Word. In Chapter 3 we add a `should_skip()` function that recognizes such files, and in Chapter 4 the real sorting plan leaves them alone.

Why the folder you run from matters

The path `data\inbox` is relative. Python resolves it against the *current working directory*, the folder your terminal is in when you start the script, not the folder the script file lives in. If you change into the `scripts` folder and run the same file, Python looks for `scripts\data\inbox`, which does not exist:

Terminal (PowerShell)

```
PS C:\CallowayTools> cd scripts
PS C:\CallowayTools\scripts> py count_inbox.py
```

Output

```
Traceback (most recent call last):
  File "C:\CallowayTools\scripts\count_inbox.py", line 9, in <module>
    for path in INBOX.iterdir():
  File "C:\Users\dana\AppData\Local\Programs\Python\Python312\Lib\pathlib.py",
line 1057, in iterdir
    for name in os.listdir(self):
                ^^^^^^^^^^^^^^^^
FileNotFoundError: [WinError 3] The system cannot find the path specified:
'data\\inbox'
```