

Don't just learn Python— build an intelligent AI companion.

Learn modern Python through hands-on AI projects and real-world applications while building your own intelligent AI companion system from scratch.

You will learn how to:

- ✓ write clean and modern Python code
- ✓ build conversational AI applications
- ✓ connect Python to modern AI models
- ✓ create memory and context-aware systems
- ✓ add voice and vision capabilities
- ✓ automate tasks with Python
- ✓ organize projects like a professional developer
- ✓ package and share your applications



About the Author

Dr. Shouke Wei is a researcher, scientist, and entrepreneur specializing in data analysis and modeling, wavelet-based signal processing, and AI-driven applications.

He earned his Ph.D. from Brandenburg University of Technology Cottbus–Senftenberg (Germany) and conducted postdoctoral research at Eawag (Switzerland). He has held research positions at the University of British Columbia (Canada) and served as a distinguished and adjunct professor at multiple universities in China.

His work focuses on bridging theoretical modeling with practical, real-world data science systems.



Learn Python by Building Your Own AI Companion SHOUKE WEI



Learn Python by Building Your Own AI Companion



Master **Modern Python** by Building a **Smart AI** Companion System



AI
Conversations



Memory &
Context



Voice &
Vision



Automation
& APIs



Real-World
Projects

SHOUKE WEI

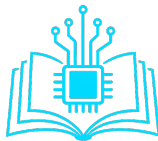
Learn Python by Building Your Own AI Companion

Master Modern Python by Building a Smart AI
Companion System

Learn Python by Building Your Own AI Companion

Master Modern Python by Building a Smart AI
Companion System

Shouke Wei



DEEPSIM
PRESS

Learn Python by Building Your Own AI Companion

Copyright © 2026 Shouke Wei
All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the author, except in the case of brief quotations used in reviews, academic citations, or other non-commercial uses permitted by copyright law.

First Edition, 2026

For permission requests, contact the publisher:
Email: shouke.wei@deepsim.ca

ISBN 978-1-0677475-1-0 (Hardcover)
ISBN 978-1-0677475-0-3 (Paperback)
ISBN 978-1-0675592-9-8 (eBook)
DOI [10.5281/zenodo.20102902](https://doi.org/10.5281/zenodo.20102902)

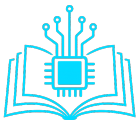
Published by

Deepsim Press

An independent imprint of Deepsim Intelligence Technology Inc.
Abbotsford, British Columbia, Canada
<https://press.deepsim.ca>

This book is intended for educational and professional readers.

For resources, updates, and companion code, visit:
<https://press.deepsim.ca/ai-companion>



DEEPSIM
PRESS

About the Author

Shouke Wei, Ph.D., is a researcher, scientist, and entrepreneur specializing in intelligent IoT systems, robotics, big data analytics, modeling and forecasting, early-warning systems, and edge computing. With academic and industry experience across Europe, North America, and Asia, Dr. Wei is recognized for bridging advanced theory with real-world, production-ready systems.

Dr. Wei earned his Ph.D. in Environmental and Resource Management from the Department of Ecosystems and Environmental Informatics at Brandenburg University of Technology Cottbus–Senftenberg (Germany). He conducted postdoctoral research at the Swiss Federal Institute of Aquatic Science and Technology (Eawag, Switzerland) and held research positions at the University of British Columbia (Canada). He has also served as a distinguished and adjunct professor at multiple institutions in China.

Dr. Wei currently serves as CEO and Chief Scientist of Deepsim Intelligent Technology Inc. (Canada) and Shandong Deepsim Intelligent Technology Co., Ltd. (China). He is also a Postdoctoral Co-Supervisor at the Shandong Postdoctoral Innovation Practice Base and Director of Qilu Artificial Intelligence and Digital Manufacturing Innovation at Shandong Deepsim Intelligent Technology Co., Ltd., China.

Dr. Wei has led or contributed to 19 major international research projects and the development of numerous intelligent systems, including autonomous water-quality monitoring vessels, AI-based environmental early-warning platforms, medical image diagnosis systems, precision agriculture robots, and autonomous service robots.

His scholarly contributions include over 40 peer-reviewed publications, 8 books on practical wavelet transform applications, data analysis, and data modeling, more than 500 technical tutorial articles, six patents, and over 30 software copyrights. His work focuses on making advanced computational methods—particularly data analysis, modeling, and wavelet-based signal processing—accessible, practical, and impactful for researchers and practitioners worldwide.

For more information, visit: <https://press.deepsim.ca/shouke/>

Contents

Preface	XV
Acknowledgments	XIX
0 Absolute Beginner Jumpstart	1
0.1 What Is a Program?	1
0.2 Why Python?	2
0.3 Install and Run Python	2
0.3.1 Install Python	3
0.3.2 Check the Installation	4
0.3.3 Try the Interactive Shell	5
0.4 Your First Program	6
0.5 What Just Happened?	6
0.6 Choose a Coding Environment	7
0.6.1 Recommended: Visual Studio Code	7
0.6.2 Beginner-Friendly Alternative: Thonny	8
0.7 Suggested Project Structure	8
0.7.1 Project Structure	8
0.7.2 Creating Project Structure with VS Code	9
0.8 Companion Resources	10
0.9 Mini Exercise	11
0.10 A Quick Note on Mistakes	11
0.11 What's Next	11
I Build Something on Day One	13
1 Your First AI Companion	15
1.1 Copy and Run the Code	15
1.2 What Every Line Does	17
1.3 Make It Yours	18
1.4 Mini Challenge	19
1.5 What's Next	19

2	From Script to Structure	21
2.1	The Problem with One Big File	21
2.2	Introducing Functions	22
2.3	Refactoring the AI Companion	23
2.4	What Changed and Why	24
2.5	Thinking in Modules	25
2.5.1	Create <code>modules/response.py</code>	26
2.5.2	Create <code>__init__.py</code>	27
2.5.3	Create <code>main.py</code>	27
2.6	Mini Challenge	29
2.7	The Core Idea	29
2.8	What's Next	30
3	Smarter Logic with a Command System	31
3.1	The Problem with Long <code>if/elif</code> Chains	31
3.2	A Better Approach: Dictionaries and Functions	32
3.3	Building the Command System	33
3.3.1	Project Structure	33
3.3.2	The Handler Functions	34
3.3.3	The Command Map	34
3.3.4	Detecting the Command	35
3.3.5	Generating the Response	36
3.3.6	The Complete <code>commands.py</code>	36
3.4	Update <code>main.py</code>	37
3.5	How the Pieces Fit Together	39
3.6	Mini Challenge	39
3.7	What's Next	40
II	Make It Useful in the Real World	41
4	Giving Your AI Memory	43
4.1	Two Kinds of Storage	44
4.2	Choosing a File Format: JSON	44
4.3	Project Structure	45
4.4	Building the Memory Module	45
4.4.1	Loading and Saving	46
4.4.2	Getting and Setting Values	48
4.4.3	The Complete <code>memory.py</code>	48
4.5	Updating the Command System	49
4.5.1	Add the New Commands to the Map — and Fix a Keyword Collision	51
4.5.2	One More Problem: Match Order	51

4.5.3	Handle Handlers That Need Arguments	52
4.6	Run the Program	53
4.7	Mini Challenge	54
4.8	What's Next	55
5	Making Your AI Do Real Work	57
5.1	What Is Automation?	57
5.2	Two New Tools: <code>pathlib</code> and <code>shutil</code>	58
5.3	Project Structure	59
5.4	Building the Automation Module	59
5.4.1	Organise Files by Type	60
5.4.2	Rename Files in Bulk	61
5.4.3	Remove Empty Folders	62
5.4.4	The Complete <code>automation.py</code>	62
5.5	Connecting to the Command System	64
5.6	Run the Program	65
5.7	What Just Happened	66
5.8	Mini Challenge	66
5.9	What's Next	67
6	Connecting Your AI to the Internet	69
6.1	What Is an API?	69
6.2	Installing the <code>requests</code> Library	70
6.3	Understanding JSON Responses	71
6.4	Project Structure	72
6.5	Building the API Module	72
6.6	Extracting the City from User Input	74
6.7	Run the Program	75
6.8	How the Full Request Cycle Works	75
6.9	Mini Challenge	76
6.10	What's Next	77
III	Add Real AI Power	79
7	Connecting Your Companion to a Real AI Model	81
7.1	Rule-Based vs AI-Generated Responses	82
7.2	Getting an API Key	82
7.2.1	Storing the Key as an Environment Variable	83
7.3	Installing the OpenAI Library	86
7.4	Project Structure	86
7.5	Your First AI Call	87
7.6	Letting the User Talk to the Model	88

7.7	Giving Your Companion a Personality	88
7.8	Organising the AI Call Into a Module	89
7.9	Saving the Conversation	90
7.10	Building the Full Companion Loop	91
7.11	Giving the AI Memory of the Conversation	93
7.12	The Complete <code>modules/ai.py</code>	94
7.13	Troubleshooting Common Problems	96
7.14	Mini Exercise	98
7.15	Challenge Project: AI Study Companion	98
7.16	What's Next	99
8	Giving Your AI a Personality	101
8.1	What Personality Means for an AI	102
8.2	Anatomy of a Good System Message	102
8.3	Project Structure:	104
8.4	Experimenting with Different Personalities	104
8.4.1	Calm Tutor	105
8.4.2	Practical Coding Coach	105
8.4.3	Encouraging Study Partner	105
8.5	Building a Proper Multi-Turn Conversation	107
8.6	Keeping History from Growing Too Long	109
8.7	Updating <code>modules/ai.py</code>	110
8.8	Mini Exercise	112
8.9	What's Next	114
9	Making Your AI Talk	115
9.1	How Voice Interaction Works	115
9.2	Installing the Libraries	116
9.3	Capturing Speech From the Microphone	118
9.4	Giving Your Companion a Voice	120
9.5	Building the Voice Module	122
9.6	Combining Voice With AI	125
9.7	The Voice Conversation Loop	126
9.8	Designing Responses for the Ear	127
9.9	Text-Input Fallback	128
9.10	Mini Exercise	128
9.11	Challenge Project: Voice Study Buddy	129
9.12	Going Further: Real-Time Streaming Voice	130
9.13	What's Next	131

IV	Make It Smarter	133
10	Emotion and Context Awareness	135
10.1	Two Kinds of Awareness	136
10.2	Project Structure:	137
10.3	Building the Emotion Module	137
10.3.1	Detecting Emotion	137
10.3.2	Adapting the Response	139
10.3.3	The Complete <code>emotion.py</code>	139
10.4	Integrating Emotion Into the Response Pipeline	140
10.5	Building Context Tracking	141
10.6	Using Context in the AI Response	142
10.7	The Full Awareness Pipeline	145
10.8	Mini Exercise	145
10.9	What's Next	147
11	Giving Your AI Eyes	149
11.1	How Computer Vision Works	150
11.2	Installing OpenCV	150
11.3	Project Structure	151
11.4	Building the Vision Module	151
11.4.1	Opening the Camera	152
11.4.2	Showing the Camera Stream	152
11.4.3	Detecting Faces	154
11.4.4	Detecting Brightness	156
11.4.5	Capturing a Single Frame for Analysis	156
11.5	Describing the Scene	157
11.5.1	The Complete <code>vision.py</code>	160
11.6	Wiring Vision Into the Command System	164
11.7	Testing Without a Webcam	166
11.8	The Vision Pipeline	167
11.9	Mini Exercise	168
11.10	What's Next	169
V	Build It Like a Real System	171
12	Organising Code Like a Professional	173
12.1	What Goes Wrong Without Structure	174
12.2	The Final Project Structure	174
12.3	Separation of Concerns	175
12.4	Creating <code>config.py</code>	177
12.5	Clean Imports	179

12.6 Reusable Utility Functions	180
12.7 Keeping Functions Small	181
12.8 Adding Proper Logging	183
12.9 The <code>requirements.txt</code> File	184
12.10 Avoiding Over-Engineering	185
12.11 Mini Exercise	186
12.12 What's Next	186
13 Async and Performance	189
13.1 Sequential vs Asynchronous Execution	190
13.2 The Key Words: <code>async</code> , <code>await</code> , and <code>asyncio.run</code>	191
13.3 Where Async Helps in Your Project	193
13.4 Making the AI Call Async	193
13.5 Updating the Command System	196
13.6 Updating <code>main.py</code>	198
13.7 Running Two Tasks at Once	200
13.8 The Complete <code>modules/api.py</code>	201
13.9 The <code>speak_async</code> Problem — and the Fix	203
13.10 When Not to Use Async	204
13.11 Mini Exercise	205
13.12 What's Next	206
14 Packaging and Sharing Your App	207
14.1 The Goal: One-Command Setup	207
14.2 Managing API Keys Safely with <code>.env</code>	208
14.3 The <code>.gitignore</code> File	209
14.4 Updating <code>requirements.txt</code>	210
14.5 Writing a <code>README.md</code>	211
14.6 Virtual Environments	213
14.7 Creating Startup Scripts	215
14.7.1 macOS and Linux	215
14.7.2 Windows	215
14.8 Sharing on GitHub	216
14.9 Test Like a New User	217
14.10 The Final Project Layout	218
14.11 Mini Exercise	219
14.12 What's Next	220
VI Final Project	221
15 Build Your Complete AI Companion	223
15.1 The Complete System at a Glance	224

15.2	The Final <code>main.py</code>	225
15.3	The Full System Flow	227
15.4	Complete System Test	228
15.4.1	Text conversation	228
15.4.2	Persistent memory	228
15.4.3	Emotion awareness	229
15.4.4	Context tracking	229
15.4.5	Weather API	229
15.4.6	File automation	230
15.4.7	Vision (if webcam available)	230
15.4.8	Voice mode (if microphone available)	230
15.5	How the Modules Connect	230
15.6	Making It Your Own	231
15.6.1	Change the Personality	231
15.6.2	Expand the Memory	232
15.6.3	Add New Commands	233
15.6.4	Improve Voice Interaction	233
15.7	What You Have Actually Learned	234
15.8	Your Path Forward	234
15.9	Final Message	235
	Further Reading	237
	Appendix A — Python Quick Reference	243
	Appendix B — Common Errors and Fixes	257
	Appendix C — Recommended Tools and Libraries	273
	Appendix D — Next Steps	283
	Index	293

Preface

Programming books often follow a familiar path: they start with syntax, move through isolated concepts, and end with small exercises.

This book takes a different approach.

From the very beginning, you will build something real.

Why This Book Exists

Python is one of the most popular programming languages in the world. There are countless tutorials, courses, and books.

Yet many learners still struggle:

- They learn syntax but can't build systems
- They complete exercises but lack real-world experience
- They feel overwhelmed when starting actual projects

This book is designed to solve that problem.

Instead of teaching Python as a collection of features, it teaches Python as a tool for **building intelligent systems**.

What You Will Build

Over fifteen chapters, you will build a complete, working AI companion — from a single `print("Hello, Python!")` to a voice-enabled, memory-persistent, emotion-aware application that connects to real AI models and runs asynchronously.

Along the way, you will:

- write Python programs that grow from a single script to a modular, multi-file system
- connect your companion to the OpenAI API and give it a custom personality
- build a persistent memory system that survives between sessions
- automate real file-system tasks with a few lines of code
- fetch live data from web APIs
- detect emotion in text and adapt your companion's responses accordingly
- capture audio from a microphone and play spoken replies through a speaker
- access a webcam and analyse what the camera sees
- structure, package, and share a professional-quality Python project

By the final chapter, you will not just have learned Python. You will have built something that demonstrates real-world skills — something you can show, share, and extend.

Beyond This Book

The system you build here is only the beginning.

You can extend it into:

- a personal assistant
- a productivity tool
- a web application
- an AI-powered product
- a physical robot (like the XiaoDi companion)

This book gives you the foundation.

What you build next is up to you.

Who This Book Is For

This book is designed for:

- absolute beginners who has little or no prior programming experience
- beginners who want to learn by building something meaningful with it
- learners who know some programming but feel stuck
- developers who want to quickly move into AI applications

This book assumes you know how to use a computer, open a terminal, and follow instructions — but no prior programming or AI experience is required. Every concept is explained clearly before it appears in code.

If you already know some Python, you can move quickly through the early chapters and focus on AI, async programming, vision, and packaging. If you are new, type the examples yourself instead of copying and pasting — the practice matters.

Most importantly, be willing to experiment and learn through trial and error. If you can think logically and keep trying, you can complete this book.

How to Use This Book

Read it in order — each chapter builds on the previous one, and later code depends on earlier concepts.

Type the code — type the code yourself instead of copying and pasting. Writing the code helps you notice details, build familiarity, and learn faster.

Break things and fix them - when errors happen, read the error messages carefully before searching for answers. Python usually tells you exactly what went wrong and where.

Build your own features - complete the exercises and experiment with your own features. Small modifications and hands-on practice are where real learning happens.

This book uses Python 3.14 and the OpenAI API with the gpt-4.1-mini model, but newer versions should work with little or no change. Check the companion repository for updates.

Final Thought

Learning programming is not about memorizing syntax.

It is about creating something meaningful.

By the time you finish this book, you won't just know Python.

👉 You will have built something real.

Shouke Wei, PhD

Deepsim Intelligent Technology Inc.

Deepsim Academy

Abbotsford, Canada

May 8, 2026

Acknowledgments

This book was inspired and shaped by many years of teaching Python programming, data analysis, modeling, and wavelet transforms. I would like to sincerely thank all of my students whose curiosity, questions, feedback, and enthusiasm continuously motivated me to improve the way I teach and explain complex ideas. Many of the practical examples and teaching approaches in this book were refined through real classroom and project experiences.

I am deeply grateful to the open-source community and the developers behind Python and its ecosystem. Their contributions have made modern programming, artificial intelligence, data science, and automation more accessible to learners around the world.

I would also like to thank the researchers, educators, and innovators working in artificial intelligence, data science, computer vision, speech technology, and intelligent systems. Their work continues to inspire new possibilities for teaching and building real-world AI applications.

Most importantly, I would like to thank my family for their support, patience, encouragement, and understanding throughout the writing of this book and many related projects. Their support made this journey possible.

Finally, thank you to every reader and learner who chooses to build, explore, and create with technology. I hope this book helps you move beyond learning syntax and toward building meaningful and intelligent systems.

0 Absolute Beginner Jumpstart

This chapter gets you from zero to your first working Python program — quickly and confidently.

By the end, you will:

- understand what Python is and why it matters
- install Python on your computer
- write and run your very first program
- choose a comfortable coding environment
- Structure the project
- learn programming in the code environment

The goal is simple: **build first, learn as you go**. Do not worry about understanding everything right away. Just follow the steps.

0.1 What Is a Program?

A **program** is a set of instructions you give to a computer. The computer follows them one step at a time, in order, and produces a result.

Think of it like a recipe:

- A recipe says: *crack two eggs, add flour, mix, bake.*
- A program says: *get the user's name, say hello, wait for input.*

The computer does exactly what you tell it — no more, no less. Your job as a programmer is to write those instructions clearly.

0.2 Why Python?

Python is one of the most popular programming languages in the world. It is used in:

- **data science** — analysing large amounts of information
- **AI and machine learning** — teaching computers to make decisions
- **automation** — making repetitive tasks run themselves
- **web development** — building websites and online tools

Python is a great first language because:

- its syntax reads almost like plain English
- it has a huge collection of ready-made libraries
- it has a large, friendly community
- errors are usually easy to understand and fix

You do **not** need any prior programming experience. If you can follow steps and try things out, you can learn Python.

0.3 Install and Run Python

This book uses the latest version available at the time of writing, Python 3.14.4. You can install any version of Python 3.12 or later depending on your computer and operating system.

0.3.1 Install Python

1. Go to <https://www.python.org/downloads/>
2. Download **Python 3.12** or a newer version
3. Run the installer
4. ✓ On Windows, tick “**Add Python to PATH**” before clicking Install — this is important (Figure 1)
5. Click **Install**

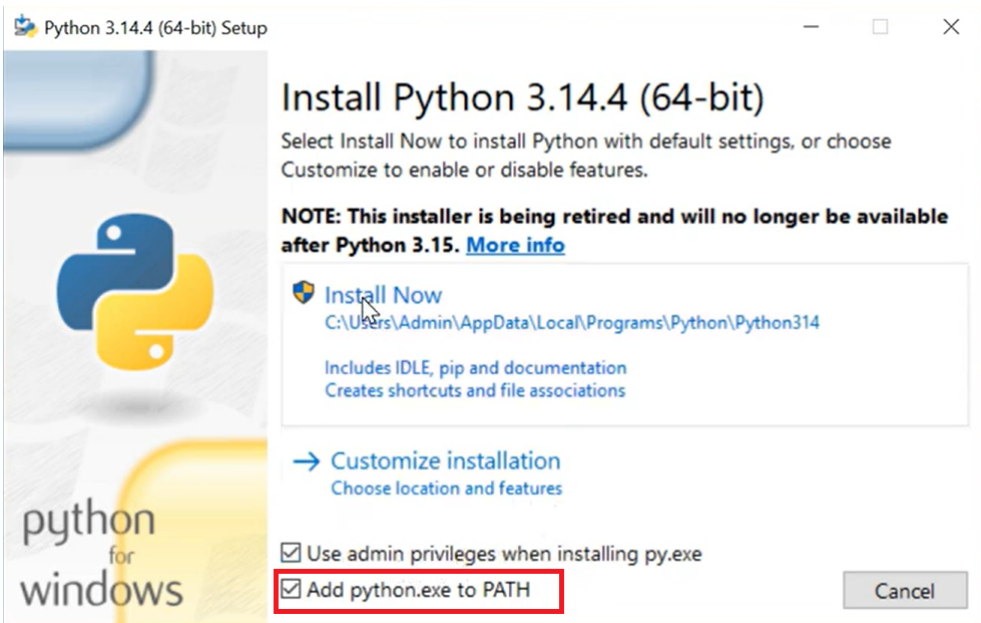


Figure 1: Screenshot of the Python installation process, highlighting the option to tick “Add Python to PATH” before clicking Install Now.

💡 Tip

If you are on macOS or Linux, Python may already be installed. Open a terminal and run `python3 --version` to check.

0.3.2 Check the Installation

Open a terminal window — this is called **Command Prompt** on Windows or **Terminal** on macOS and Linux — and type:

```
python --version
```

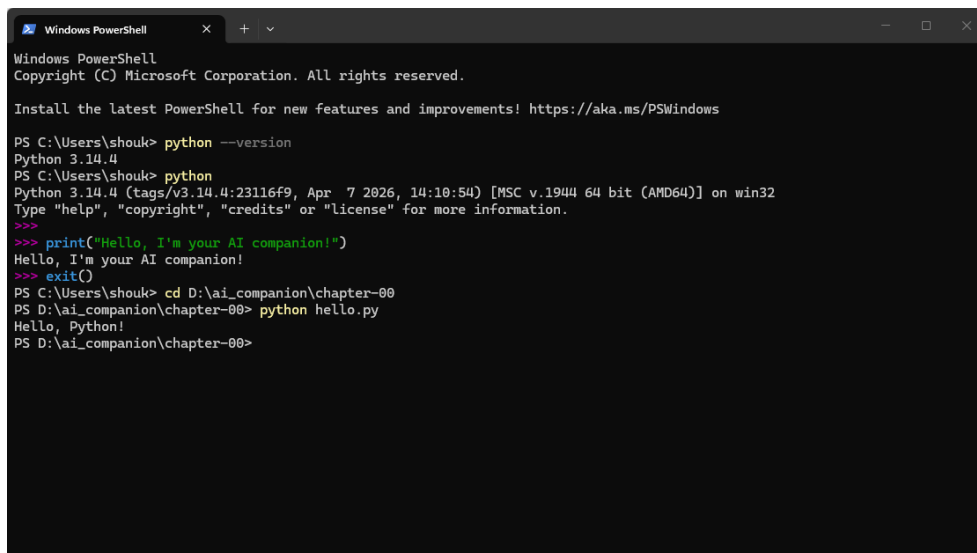
You should see something like (Figure 2):

```
Python 3.1x.x
```

For example, on my computer:

```
Python 3.14.4
```

If you see that, Python is installed and ready. If you see an error, revisit step 4 and make sure you ticked “Add Python to PATH”.



```
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\shouk> python --version
Python 3.14.4
PS C:\Users\shouk> python
Python 3.14.4 (tags/v3.14.4:23116f9, Apr 7 2026, 14:10:54) [MSC v.1944 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print("Hello, I'm your AI companion!")
Hello, I'm your AI companion!
>>> exit()
PS C:\Users\shouk> cd D:\ai_companion\chapter-00
PS D:\ai_companion\chapter-00> python hello.py
Hello, Python!
PS D:\ai_companion\chapter-00>
```

Figure 2: Screenshot of running Python, including checking the Python version, using the interactive shell, exiting Python, and running a Python program.

0.3.3 Try the Interactive Shell

Type this in your terminal and press Enter:

```
python
```

You will see a prompt that looks like this (Figure 2):

```
>>>
```

This is the **Python interactive shell** (also called a REPL — Read, Evaluate, Print, Loop). You can type Python code here and see the result immediately. It is a great way to experiment.

Type the following and press Enter (Figure 2):

```
print("Hello, I'm your AI companion!")
```

Output:

```
Hello, I'm your AI companion!
```

`print()` is one of the most commonly used functions in Python. It takes whatever you put inside the parentheses and displays it on the screen.

To leave the interactive shell, type `exit()` and press Enter (Figure 2).

0.4 Your First Program

The interactive shell (Section 0.3.3) is useful for quick experiments, but it does not save your work. When you close it, everything is gone. The standard way to write programs is to save your code in a **file**.

Python files end with `.py`. Let's create your first one.

Open any plain text editor (Notepad on Windows works fine for now) and type:

```
print("Hello, Python!")
```

Save the file as `hello.py`.

Now, back in your terminal, navigate to the folder where you saved the file (e.g. `D:\ai_companion\chapter-00`, see Section 0.7) and run (see Figure 2):

```
python hello.py
```

You should see:

```
Hello, Python!
```

Congratulations — you have just written and run your first Python program.

0.5 What Just Happened?

Here is the core idea of every program you will ever write:

👉 **Write code** → **Run it** → **See the output**

When you ran `python hello.py`, Python read your file line by line, followed your instructions, and printed the text to the screen. Everything else you learn in this book builds on that simple loop.

0.6 Choose a Coding Environment

A plain text editor works, but a proper **code editor** or **IDE** (Integrated Development Environment) makes your life much easier. IDEs offer features like:

- colour-coding that makes code easier to read (syntax highlighting)
- automatic suggestions as you type (code completion)
- built-in tools for finding and fixing errors (debugging)

You only need to learn the basics right now: how to create a file, write code, and run it.

0.6.1 Recommended: Visual Studio Code

Visual Studio Code (VS Code) is free, beginner-friendly, and used by professionals worldwide. It works on Windows, macOS, and Linux.

1. Download and install VS Code: <https://code.visualstudio.com/>
2. Open VS Code
3. Click the **Extensions** icon like four squares in the left sidebar (Figure 3)
4. Search for **Python** and install the extension published by Microsoft

That is all you need for now. Do not worry about any other settings.

The screenshot below shows the VS Code interface with the key areas labelled:

Enable Auto-Save

To avoid manually saving files each time, enable Auto-Save in VS Code by clicking **File > Auto Save**. Once checked, your files will be saved automatically as you type.

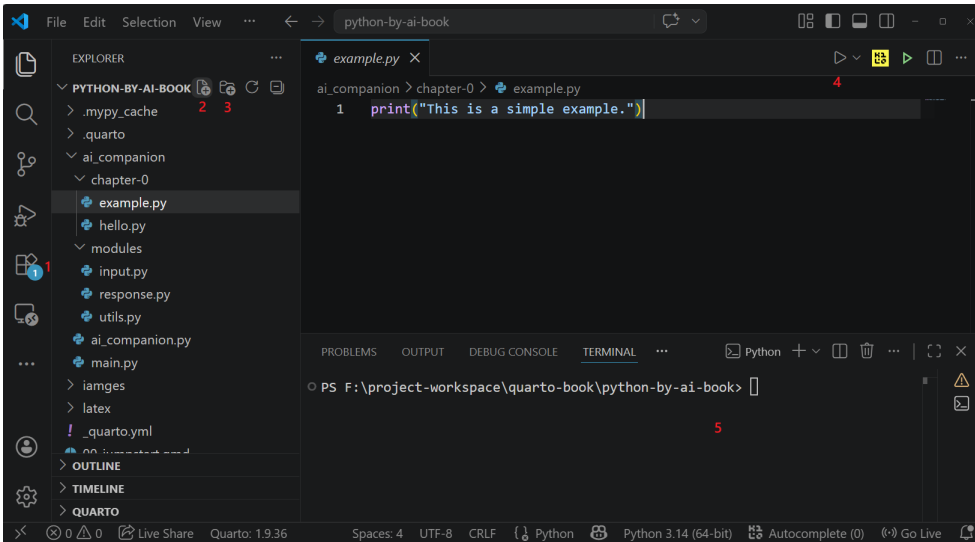


Figure 3: Screenshot of VS Code interface with the key areas labelled: 1. Extensions icon, 2. New file button, 3. Project folder panel, 4. Run Python file button, and 5. Integrated terminal.

0.6.2 Beginner-Friendly Alternative: Thonny

If VS Code feels overwhelming, try **Thonny** (<https://thonny.org>). It is designed specifically for beginners, has a very simple interface, and comes with Python bundled — so there is less to install.

0.7 Suggested Project Structure

0.7.1 Project Structure

As you work through this book, it helps to keep your files organised. Here is a simple folder structure to follow:

```
ai_companion/  
├── chapter-00/  
│   ├── hello.py  
│   └── example.py  
├── chapter-01/  
└── ai_companion.py
```

```
├─ chapter-02/
│  └─ main.py
│     └─ modules/
│        └─ input.py
│           └─ response.py
│              └─ utils.py
└─ ...
```

Create a folder called `ai_companion` somewhere on your computer. Inside it, create a sub-folder for each chapter as you go. This keeps everything tidy and makes it easy to find your work later.

0.7.2 Creating Project Structure with VS Code

VS Code makes it easy to set up your project structure (see Figure 4) and for the process using buttons (see Figure 3).

1. Open VS Code and click **File > Open Folder...** to open the `ai_companion` folder you created.
2. Click **Terminal > New Terminal** to open an integrated terminal.
3. In the terminal, type `mkdir chapter-00` to create the Chapter 0 folder — or use the **New Folder** button in the Explorer panel.
4. Navigate into the folder with `cd chapter-00`.
5. Create a new Python file by typing `code hi.py` in the terminal — or click the **New File** button in the Explorer panel.
6. Save the file via **File > Save**, or press `Ctrl+S`.
7. Run the `.py` file directly in the VS Code terminal by typing `python hi.py`, or click the **Run Python File** button in the top-right corner when the file is open in the editor.
8. Type `cd ..` to return to the parent folder `ai_companion`, then repeat the process to create folders for the remaining chapters.

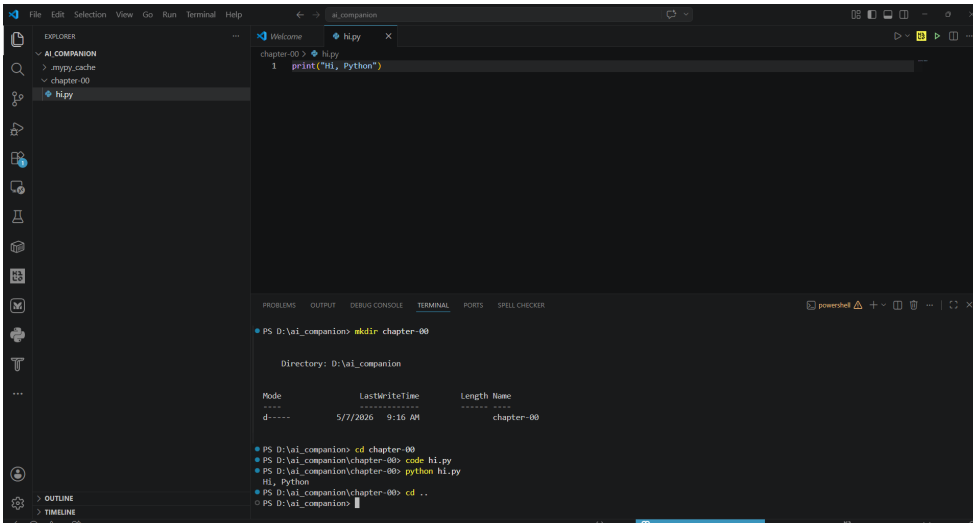


Figure 4: Screenshot showing the process of using VS Code to create a project structure, navigate into a folder, create and run `.py` files, and return to the parent folder.

0.8 Companion Resources

The structured folders and scripts used in this book are available online.

Official resource hub: <https://press.deepsim.ca/ai-companion>

GitHub repository (code and reproducible workflows): <https://github.com/shoukewei/ai-companion>

The repository contains:

- all structured folders used throughout the chapters
- all scripts developed throughout the chapters
- reproducible examples and workflows
- supporting project files and configurations

0.9 Mini Exercise

Open `hello.py` in your editor and replace the contents with this:

```
print("Hello, AI!")  
print("I am learning Python.")
```

Save the file in the `chapter-00` folder, and run it again:

```
python hello.py
```

You should see two lines of output. Notice that each `print()` call produces one line. You are already writing real programs.

0.10 A Quick Note on Mistakes

You will make mistakes. Every programmer does — including experienced ones. When Python shows an error message, it is not scolding you. It is giving you a clue about what went wrong and where.

Read the error message carefully. It usually tells you the line number and the type of problem. You will learn to read these messages quickly, and they will become your best debugging tool.

0.11 What's Next

Many beginners stop before they start because setup feels confusing. You just crossed that barrier.

In the next chapter, you will build your first AI companion in under 30 minutes — no theory overload, just building.

👉 Keep going. The best way to learn Python is to write Python.

Part I

Build Something on Day One

1 Your First AI Companion

In this chapter you will build your **first working AI companion** — a real program that listens to what you type and responds intelligently. You will have it running in about 30 minutes.

By the end, your program will behave like this:

```
AI Companion started. Type 'quit' to exit.
```

```
You: hello
```

```
AI: Hi! Nice to meet you.
```

```
You: how are you
```

```
AI: I'm doing great. Thanks!
```

```
You: quit
```

```
AI: Goodbye!
```

It is simple — and that is the point. You will understand every line of code before the chapter is done, and you will know exactly how to extend it.

1.1 Copy and Run the Code

Inside your `chapter-01` folder, create a new file called `ai_companion.py` and paste in the following code:

Index

Symbols

.bashrc, [83](#)
.env file, [208](#)
.gitignore, [209](#)
.py file, [6](#)
.zshrc, [83](#)
__name__, [24](#)
30-day learning plan, [290](#)

A

adapt_response function, [139](#)
adaptive responses, [135](#)
AI agents, [284](#)
AI applications, [284](#)
AI companion, [11](#)

- complete system, [220](#), [223](#)
- extending, [19](#), [98](#)
- first program, [15](#)
- personalising, [231](#)

AI models

- connecting to, [77](#)

AI personality, [88](#)
AI-generated responses, [81](#)

- advantages, [82](#)

ai.py

- async version, [193](#)
- complete file, [94](#)
- context-aware, [142](#)
- multi-turn, [110](#)

ai_companion.py, [15](#)
ai_companion_v2.py, [23](#)
any(), [62](#)
API key, [82](#)

- safe storage, [208](#)
- troubleshooting, [263](#)

api.py, [72](#)

- complete file, [201](#)

APIs, [67](#)

- definition, [69](#)
- extending, [76](#)
- introduction, [69](#)
- request-response cycle, [75](#)

architecture

- command pipeline, [39](#)

arithmetic operators, [245](#)
ask_ai function

- voice version, [125](#)

async def

- calling sync functions, [197](#)

async def ask_ai, [193](#)
async keyword, [191](#), [254](#)
async programming, [186](#), [189](#)

- common mistakes, [268](#)
- extending, [205](#)
- use cases, [193](#)
- when to avoid, [204](#)

asyncio, [189](#), [254](#)

- final loop, 225
- `asyncio.gather()`, 200, 254
- `asyncio.run()`, 122, 191, 198, 254
 - nested event loop error, 203
- `asyncio.to_thread()`, 198, 201
- `asyncio.to_thread()`, 226
- AsyncOpenAI, 193
- authentication, 82
- AuthenticationError, 263
- automation, 55, 57
 - advanced, 285
 - definition, 57
 - extending, 66
 - pipeline, 66
- `automation.py`, 59
 - complete file, 62
- `await` keyword, 191, 254
 - missing, 268

B

- batch files, 215
- blocking code, 190
- blocking operations, 193
- boolean operators, 246
- break statement, 17, 247, 265
- brightness detection, 156

C

- camera
 - opening, 152
- camera input, 149
- camera warm-up, 156
- `cap.release()`, 152
- capstone project, 220, 223
- `capture_frame`

- warm-up, 269
- `capture_frame` function, 156
- classification, 287
- `clean_empty_folders` function, 62
- client-server model, 69
- `client.responses.create()`, 87
- closing, 235
- code editor, 7
- code execution, 6
- code organisation, 169, 173
- code smell, 174
- code structure, 21
- command handling, 30
- command prompt, 4
- command system, 31
 - extending, 39
- commands
 - adding new, 233
- COMMANDS dictionary, 34
 - extending, 233
 - ordering, 51
 - updating, 51, 64
 - vision, 164
- `commands.py`
 - async version, 196
 - automation commands, 64
 - complete file, 36
 - emotion integration, 140
 - updating, 49
 - vision commands, 164
- comparison operators, 246
- computer vision, 147, 149
 - extending, 168
 - next steps, 289

- overview, 150
- concurrency, 186, 189
- concurrent execution, 200
- conditional statements, 17
 - advanced, 30
- config.py, 177
 - importing from, 178
- configuration
 - central, 177
- constants
 - configuration, 177
- context awareness, 131, 135
- context tracking, 141
- context window, 93
 - managing, 109
- continue statement, 126, 247
- continuous listening, 233
- conversation history
 - feeding to model, 93
 - message list, 107
 - saving, 90
 - trimming, 109, 268
- conversation loop, 91
- conversation topic, 141
- coroutines, 191
- count_faces_in_frame function, 157
- customisation, 18, 231
- cv2.CascadeClassifier(), 154
- cv2.imshow(), 152
- cv2.VideoCapture(), 152
- cv2.waitKey(), 152

D

- data analysis, 284
 - time series, 288

- data types, 244
- data visualisation, 284
- databases, 289
- debugging
 - checklist, 271
 - mindset, 11
- def keyword, 22, 249
- default parameters, 249
- dependencies, 184, 210
- dependency graph, 230
- deployment, 206, 207
- describe_scene function, 157
- desktop applications, 287
- detect_command function, 35
- detect_emotion function, 137
- detectMultiScale(), 154
- developer experience, 207
- dict.get(), 48, 248
- dict.items(), 248
- dict.keys(), 248
- dictionaries, 33, 248
 - command mapping, 32, 34
 - nested, 71
- dispatch table, 32
- distribution, 206, 207
- documentation, 211
- dotenv, 198
- downloads folder, 59
- DRY principle, 180

E

- edge-tts, 116, 130
 - async, 120
 - voices, 120
- elif, 17, 18, 246
 - scalability problem, 31

- else, [17](#), [246](#)
- emotion detection, [131](#), [135](#)
 - extending, [145](#)
 - vs context tracking, [136](#)
- emotion.py, [137](#)
 - complete file, [139](#)
- encoding
 - UTF-8, [90](#), [250](#), [265](#)
- enumerate(), [61](#)
- environment variables, [83](#)
 - .env file, [208](#)
 - where stored, [83](#)
- error handling
 - voice loop, [126](#)
- errors
 - API key, [96](#)
 - beginners, [11](#)
 - library version, [96](#)
- event loop, [190](#)
- exceptions, [252](#)
- exercise, [11](#), [19](#), [29](#), [39](#), [54](#), [66](#), [76](#), [98](#), [112](#), [128](#), [145](#), [168](#), [186](#), [205](#), [219](#)
- explicit imports, [179](#)

F

- f-strings, [49](#), [244](#)
- face detection, [154](#)
 - troubleshooting, [269](#)
- fallback
 - text input, [128](#)
- FastAPI, [280](#), [286](#)
- faster-whisper, [130](#)
- file path errors, [266](#)
- file system
 - automation, [57](#), [285](#)
 - interaction, [66](#)
- FileNotFoundError, [252](#), [261](#)
- files
 - appending, [90](#)
 - reading and writing, [43](#), [250](#)
 - saving data, [40](#)
- final project, [220](#), [223](#)
- Flask, [280](#), [286](#)
- Flet, [281](#), [287](#)
- float, [245](#)
- folder organisation, [8](#), [174](#)
- for loop, [247](#)
 - dictionary iteration, [35](#)
- forecasting, [288](#)
- frame
 - video frame, [150](#)
- function calling, [284](#)
- functions
 - arguments, [52](#)
 - as values, [32](#)
 - ask_ai, [89](#)
 - command handlers, [34](#)
 - definition, [22](#), [249](#)
 - generate_response, [24](#)
 - get_user_input, [24](#)
 - main, [24](#)
 - motivation, [21](#)
 - separation of concerns, [23](#)
 - size guidelines, [181](#)
- further reading, [234](#)

G

- generate_response function, [36](#)
 - emotion-aware, [140](#)
- generate_response_async function, [196](#)

- `get_input` function, 227
- `get_memory` function, 48
- getting started, 1
 - first project, 15
- Git, 281
 - basic workflow, 216
 - ignoring files, 209
- `git commit`, 216
- `git init`, 216
- GitHub, 216, 281
- GPT model, 81

H

- Haar cascade, 150
 - face detection, 154
- hard-coded values, 177
- HTTP
 - request cycle, 75
- HTTP request, 69
- HTTP response, 69
- `httpx`, 275

I

- IDE, 7, 274
- `if __name__ == "__main__"`, 24
- if statement, 17, 246
 - limitations, 31
- image recognition, 147
- import statement, 25, 37, 252
- imports
 - best practices, 179
- in operator, 17
- in-memory storage, 44
- indentation, 258
- `IndentationError`, 258
- infinite loop, 265

- `input()`, 17, 245
- `insufficient_quota`, 96
- `int`, 245
- integration, 223
- interactive AI loop, 88
- interactive shell, 5
- internet
 - connecting to, 67, 69
- isolated environment, 213

J

- JSON, 44, 251
 - API responses, 71
 - parsing, 71
 - parsing errors, 264
- `json` module, 44, 276
- `json.dump()`, 46, 251
- `json.dumps()`, 251
- `json.load()`, 46, 251
- `json.loads()`, 251
- `JSONDecodeError`, 264
- JupyterLab, 274

K

- keyword collision, 51
- keyword matching
 - emotion, 137
 - longest match first, 51

L

- learning path, 234
 - choosing, 283
- learning plan, 290
- learning strategy, 291
- `list.pop()`, 247
- `list.remove()`, 247
- lists, 247

- LLMs
 - advanced use, [284](#)
 - connecting to, [81](#)
 - introduction, [77](#)
- logging
 - configuration, [183](#)
 - levels, [183](#)
 - DEBUG, [184](#)
 - INFO, [184](#)
 - WARNING, [184](#)
- logging module, [183](#)
- M**
- machine learning, [287](#)
- main.py, [27](#)
 - async loop, [198](#)
 - final version, [225](#)
 - updating imports, [37](#)
- maintainability, [173](#)
- Markdown
 - README, [211](#)
- matplotlib, [284](#)
- memory
 - AI companion, [43](#)
 - conversation history, [40](#)
 - expanding, [232](#)
 - extending, [54](#)
- memory.py
 - complete file, [48](#)
- messages list, [107](#)
- messages[:]=, [111](#)
- microphone
 - troubleshooting, [267](#)
- ModuleNotFoundError, [262](#)
- modules
 - __init__.py, [27](#)
 - ai.py, [89](#)
 - updating, [110](#)
 - api.py, [72](#)
 - automation.py, [59](#)
 - commands.py, [33](#)
 - dependencies, [230](#)
 - emotion.py, [137](#)
 - importing, [252](#)
 - introduction, [25](#)
 - jokes.py, [29](#)
 - memory.py, [45](#)
 - context, [141](#)
 - response.py, [26](#)
 - responsibilities, [175](#)
 - summary, [224](#)
 - vision.py, [151](#)
 - voice.py, [122](#)
 - multi-turn conversation, [107](#)
 - multi-turn conversations, [99](#)
 - multimodal AI, [147](#), [149](#)
- N**
- NameError, [259](#)
- next steps, [234](#)
- nltk, [279](#)
- non-blocking code, [190](#)
- normalize_text function, [180](#)
- np.mean(), [156](#)
- numpy, [150](#), [276](#)
- O**
- one-command setup, [207](#)
- open()
 - append mode, [90](#)
- open(), [46](#), [250](#)
- OpenAI

- API key, 82
- billing, 96
- first API call, 87
- library, 275
- OpenAI API, 77, 81
- openai library, 86
- OPENAI_API_KEY, 83, 263
- OpenCV, 149, 278
 - advanced, 289
 - camera warm-up, 269
 - installation, 150
- openpyxl, 276
- organize_files function, 60
- os.environ, 83
- os.getenv(), 208
- over-engineering, 185, 204

P

- packaging, 206, 207
 - extending, 219
- pandas, 276, 284
- parallel tasks, 200
- PATH environment variable, 2
- Path object, 45, 58, 250
- Path.exists(), 46, 250
- Path.iterdir(), 60
- Path.mkdir(), 266
- Path.mkdir(), 60, 250
- Path.rename(), 61
- Path.rmdir(), 62
- Path.suffix, 60
- pathlib, 45, 58, 250, 276
- performance, 186, 189
 - troubleshooting, 268
- PermissionError, 264
- persistent storage, 44

- personality
 - AI companion, 99, 101
 - calm tutor, 105
 - coding coach, 105
 - customising, 231
 - definition for AI, 102
 - examples, 104
 - multiple modes, 112
 - study partner, 105
- personality switcher, 112
- Pillow, 278
- pip
 - installing packages, 70, 86, 116, 150, 262
 - package manager, 274
 - requirements file, 184
 - virtual environment, 213
- pip freeze, 210
- pipeline
 - complete system, 227
 - emotion and context, 145
 - vision, 167
- pixels, 150
- playsound3, 116
 - playing audio, 120
- print(), 5, 17, 243
 - replacing with logging, 183
- program
 - definition, 1
- project
 - AI study companion, 98
 - voice study buddy, 129
- project ideas, 292
 - advanced, 292
 - beginner, 292

- intermediate, 292
- project structure, 8, 255
 - final, 174, 218
 - modules, 25
- project-based learning, 291
- prompt engineering, 101
 - advanced, 284
 - experimentation, 104
 - system message, 88
 - voice-optimised, 127
- PyAudio, 276
 - installation, 267
- PyCharm, 274
- PySide6, 281, 287
- Python
 - core concepts, 19
 - dictionaries, 33
 - installation, 2
 - introduction, 1
 - running a script, 6, 15
 - uses, 2
 - version, 273
 - what you have learned, 234
 - why learn, 2
- python-dotenv, 208, 274
- pyttsx3 library, 276

Q

- Quarto, 281

R

- range(), 247
- RateLimitError, 96
- README.md, 211
 - purpose, 207
- real-time data, 69

- real-world tasks, 55, 57
- recent memory, 93
- recognizer.recognize_google(), 118
- refactoring, 21, 23, 89, 173
 - extract function, 181
 - project, 186
- regression, 287
- rename_files function, 61
- repetitive tasks, 57
- REPL, 5
- reproducibility, 217
- requests library, 70, 275
- requirements.txt, 184
 - finalising, 210
- retrieval-augmented generation, 284
- return statement, 22, 249
 - multiple values, 24
- roles
 - assistant, 107
 - system, 89, 107
 - user, 89, 107
- rule-based systems
 - limitations, 82
- running the program, 53, 65, 75

S

- scalability, 31, 169
- scene analysis, 156
- scene description, 157
- schedule library, 279
- scheduling, 285
- scikit-learn, 287
- scipy, 116
- send_output function, 227

- sensitive files, 209
- sentiment analysis, 131, 135
- separation of concerns, 175
- sequential execution, 190
- serialisation, 44
- set_memory function, 48
- setx command, 83
- sharing projects, 207
- shell scripts, 215
- shutil, 58, 279
- shutil.move(), 58
- simplicity, 185
- single frame capture, 156
- single responsibility principle, 29, 175
 - functions, 181
- skills acquired, 234
- slicing, 247
 - message list, 109
 - string, 93
- software architecture, 169, 173
- software design
 - beginner, 21
 - command pattern, 31
- sorted(), 51
- sounddevice, 116
 - recording, 118
- soundfile
 - saving wav, 118
- spaghetti code, 21, 174
- speak_async function, 226
- speak_async(), 203
- speech recognition
 - microphone input, 118
 - overview, 115
 - speech-to-text, 114, 115
 - SpeechRecognition
 - troubleshooting, 267
 - SpeechRecognition library, 116, 276
- SQL, 289
- SQLite, 289
- sqlite3, 289
- sr.AudioFile(), 118
- sr.RequestError, 118
- sr.UnknownValueError, 118
- start.bat, 215
- start.sh, 215
- starter setup, 282
- startup scripts, 215
- stateful programs, 43
- static image
 - vision testing, 166
- str.lower(), 17, 244
- str.replace(), 74, 244
- str.split(), 49, 244
- str.strip(), 49, 74, 244
- streaming voice assistant, 130
- Streamlit, 280, 286
- string manipulation
 - city extraction, 74
- strings, 244
- SyntaxError, 257
- system architecture
 - flow, 227
 - overview, 224
- system message, 88
 - context, 102
 - context injection, 142
 - customising, 231

- designing, 101
- dynamic, 144
- examples, 104
- purpose, 102
- role, 102
- rules, 102
- structure, 102
- study companion, 98
- tone, 102
- voice companion, 127

system prompts

- advanced, 99

system test

- scenarios, 228

T

- technical debt, 174
- terminal, 4
- terminal commands, 255
- test folder, 59
- testing
 - clean install, 217
 - complete system, 228
 - without webcam, 166
- text-to-speech, 114, 115
 - edge-tts, 120
 - overview, 115
- TextBlob, 145, 279
- Thonny, 8, 274
- time series, 288
- Tkinter, 281
- token limits, 109
- tone adaptation, 139
- topic detection, 142
- troubleshooting
 - OpenAI, 96

- try/except, 52, 252
- tuples
 - returning multiple values, 25
- TypeError, 52, 259
 - count_faces_in_frame, 270

U

- underscore variable, 38
- UnicodeDecodeError, 265
- update_context function, 142
- user input
 - AI model, 88
- user testing, 217
- utility functions, 180
- utils.py, 180

V

- VADER sentiment analyser, 145, 279
- ValueError, 252, 260
- variables, 244
 - lifetime, 44
- venv, 213, 253
- version control, 216
- virtual environment, 213, 253, 274
 - package not found, 267
- vision.py, 151
 - complete file, 160
- Visual Studio Code, 7, 274
- voice assistant, 115
- voice design, 127
- voice interaction, 114, 115
 - async event loop, 203
 - combined with AI, 125
 - conversation loop, 126

- extending, [128](#)
- fallback, [128](#)
- improving, [233](#)
- pipeline, [115](#)
- study buddy, [129](#)

[voice.py](#), [122](#)

`VOICE_MODE`, [225](#)

VS Code, [7](#)

W

wake phrase, [126](#)

wake word, [233](#)

web development, [286](#)

web requests, [67](#), [69](#)

web scraping, [285](#)

while loop, [17](#), [247](#)

- conversation loop, [91](#)

- infinite, [265](#)

wildcard imports, [179](#), [252](#)

Windows Registry, [83](#)

with statement, [46](#), [250](#)

wtr.in API, [72](#)

Y

YAGNI principle, [185](#)