

PYTHON WORKSHOP / FREE FOUNDATIONS

SET UP & LEARN PROGRAMMING THE RIGHT WAY

Build your coding setup. Learn the workflow. Start with the right habits.

PRE-ISSUE 00C

34-PAGE FOUNDATION

FREE EDITION

THE GOAL

Know what to install, where to write code, how the pieces connect, and why a computer is the recommended primary learning environment.

COMPUTER-FIRST / BEGINNER SETUP / DIGITAL EDITION / BY UPGRADEFEELING

YOUR BEGINNER CODING SETUP

PRE
00C

COMPUTER

Your primary workspace



PYTHON

The language + interpreter



EDITOR / IDE

Where you work with source files



TERMINAL

Where commands and programs can run



PROJECT FOLDER

Where your work stays organized

UF That Upgrade Feeling
upgradefeeling.com

FRONT MATTER

FREE EDITION

A free guide to getting your programming environment ready

Before the first real Python project, learn what each tool does, how the pieces connect, and how to verify that your setup actually works.

FREE FOUNDATIONS SERIES

PRE-ISSUE 00C · SET UP & LEARN PROGRAMMING THE RIGHT WAY · @UPGRADEFEELING

WHAT THIS GUIDE WILL DO

- Help you choose a practical beginner setup.
- Explain editor, IDE, interpreter, terminal, files, and paths.
- Walk through installation and verification.
- Show how to use AI without outsourcing the thinking.

WHAT IT WILL NOT DO

- Turn setup into a tool-shopping contest.
- Require a high-end development machine.
- Teach paid-issue Python syntax early.
- Pretend one editor is universally best.

VERSION-AWARE BY DESIGN

Installer screens and product interfaces change. This guide teaches the stable concepts first, then uses current official setup guidance where exact steps matter.

FRONT MATTER

WELCOME

Build the setup before you build the habit

A clean setup removes avoidable friction. It does not teach programming for you - but it gives you a reliable place to practice, test, organize, and debug your work.

FOUNDATION PRINCIPLE

Use a computer as your primary programming environment whenever you reasonably can.

Phones and tablets can run code in some situations. The recommendation is about workflow quality, not about whether mobile coding is technically possible.

THE FOUR THINGS YOU SHOULD UNDERSTAND

WHERE CODE LIVES

01

WRITE

Editor / IDE

WHAT RUNS IT

02

RUN

Interpreter

HOW YOU VERIFY

03

CONTROL

Terminal / Run tools

HOW YOU ORGANIZE

04

STORE

Project folder

THE TOOL IS NOT THE SKILL

VS Code, PyCharm, a terminal, and Python are parts of the environment. Programming skill comes from understanding problems, code, behavior, tests, and corrections.

FRONT MATTER

READER GUIDE

Do. Verify. Understand. Continue.

Do not treat setup as a sequence of blind clicks. After each step, verify the result and understand what changed.

THE SETUP CYCLE

- 1 DO THE STEP**
Install, create, select, or open the required tool.
- 2 VERIFY IT**
Run the smallest check that proves the step worked.
- 3 UNDERSTAND THE ROLE**
Know what that tool does - and what it does not do.
- 4 CONTINUE ONLY WHEN READY**
Fix setup problems before adding another layer.

YOUR STUDY CHECK

- ☐ I know what each installed tool is for.
- ☐ I verify setup steps instead of assuming they worked.
- ☐ I keep my project files in a known folder.
- ☐ I use help without skipping the reasoning.

INSTALLATION IS NOT COMPLETE UNTIL YOU VERIFY IT

A successful installer window is not the same thing as a working programming environment.

FRONT MATTER

CONTENTS

Pre-Issue 00C at a glance

Thirty-four pages take you from device choice to a verified local Python workspace, then into the habits that make that workspace useful.

GET READY

01-08

- | | |
|---|---|
| 01 Cover | 02 Free Edition |
| 03 Welcome - Build the Setup Before the Habit | 04 How to Use - Do. Verify. Understand. Continue. |
| 05 Contents | 06 What You Actually Need to Start |
| 07 Computer, Tablet or Phone? | 08 When Mobile Coding Makes Sense |

CHOOSE YOUR TOOLS

09-13

- | | |
|---|------------------------------|
| 09 What Is a Code Editor? | 10 What Is an IDE? |
| 11 Editor vs IDE vs Interpreter vs Terminal | 12 Which One Should You Use? |
| 13 Do Not Waste a Week Choosing an Editor | |

INSTALL & CONNECT

14-20

- | | |
|----------------------------------|----------------------------------|
| 14 Install Python - Windows | 15 Install Python - macOS |
| 16 Install Python - Linux | 17 Verify Python |
| 18 Install Visual Studio Code | 19 Add Python Support to VS Code |
| 20 Select the Python Interpreter | |

FILES & WORKFLOW

21-28

- | | |
|---|----------------------------------|
| 21 Open Your Workshop Folder in VS Code | 22 Create Your First Python File |
| 23 Run Python from VS Code | 24 Meet the Terminal |
| 25 Run Python from the Terminal | 26 Run Button vs Terminal |
| 27 What Is a Debugger? | 28 Extensions vs Python Packages |

LEARN WELL & START

29-34

- | | |
|--|---|
| 29 Local vs Browser-Based Coding | 30 Why We Recommend a Computer |
| 31 Use AI Without Giving Away the Thinking | 32 Setup Checklist |
| 33 Troubleshooting Before You Panic | 34 You Are Ready - Continue to Issue 01 |

WORKSHOP BASELINE

COMPUTER + PYTHON + EDITOR / IDE + TERMINAL + ORGANIZED PROJECT FOLDER

GET READY - BEGINNER SETUP

PAGE 06

What do you actually need to start programming?

For beginner Python work, the setup is smaller than it looks. You need a computer, a Python interpreter, a place to edit source files, and a way to run and verify them.

COMPUTER

A laptop or desktop gives you the normal development workflow: files, folders, keyboard, terminal, multiple windows, and local tools.

PYTHON INTERPRETER

The software that runs Python programs. Installing an editor does not automatically install Python itself.

EDITOR OR IDE

The place where you create and edit source files. Different tools can fill this role.

RUN / TERMINAL WORKFLOW

A way to issue commands, run a file, inspect output, and verify what happened.

YOU DO NOT NEED A HIGH-END MACHINE

The console exercises in Python Workshop are intentionally lightweight. The goal is a stable workspace, not expensive hardware.

INTERNET HELPS WITH SETUP

You need it to download tools and access current documentation. Once installed, many beginner local exercises can run without a constant connection.

NEXT: DEVICE CHOICE

CAN A PHONE DO THIS? SOMETIMES. SHOULD IT BE YOUR PRIMARY WORKSPACE? USUALLY NOT.

GET READY - DEVICE CHOICE

PAGE 07

Computer, tablet, or phone?

You can write and run code on more than one kind of device. The better question is which device gives you the full workflow you are trying to learn.

RECOMMENDED PRIMARY

COMPUTER

- Full physical or external keyboard.
- Normal access to files and project folders.
- Terminal and debugger workflows.
- Multiple files and windows at once.
- Better support for local tools and version control later.

vs

POSSIBLE IN SOME CASES

PHONE / TABLET

- Quick snippets and small exercises.
- Browser-based coding environments.
- Reviewing code or documentation.
- Convenient when a computer is unavailable.
- Workflow limitations vary by app, OS, and service.

WORKSHOP RECOMMENDATION

POSSIBLE ≠ RECOMMENDED

Mobile coding is real. Python Workshop is still designed computer-first because the environment itself is part of what you are learning.

THIS IS A WORKFLOW RECOMMENDATION - NOT A HARDWARE RULE

If a phone or tablet is all you have, you can still practice. Use what is available, then move to a computer when the task starts requiring richer file, terminal, debugging, or project workflows.

GET READY - MOBILE CODING

PAGE 08

When does mobile coding make sense?

A phone is useful when the task is small and self-contained. It becomes less practical when the learning goal includes the surrounding development environment.

GOOD FIT**Quick experiments**

Try a short expression, tiny snippet, or compact challenge when you want immediate practice.

GOOD FIT**Review and reading**

Read code, inspect a solution, review notes, or check documentation away from your main computer.

GOOD FIT**Browser playgrounds**

Useful for self-contained exercises when you do not need a complete local project setup.

MOVE TO A COMPUTER**Project workflow**

Multiple files, local tooling, terminal work, debugging, Git, packages, and larger projects are usually easier there.

QUICK CHECK

You can run a small snippet on your phone. The next exercise asks you to manage several files, use a terminal, and inspect a debugger. Which should be your primary workspace?

Stay on the phone because it can run code.

Use a computer for the fuller workflow.

The device never affects the learning workflow.

Choose an answer.

NEXT: YOUR WRITING TOOL

PAGE 09 EXPLAINS WHAT A CODE EDITOR ACTUALLY IS - BEFORE WE COMPARE IT WITH AN IDE.

GET READY - CODE EDITOR

PAGE 09

What is a code editor?

A code editor is software for creating and changing source-code files. It gives plain text a programming-friendly workspace without becoming the programming language itself.

PLAIN-TEXT SOURCE

Edit the actual file

The editor works with source files such as .py. The code remains text that other tools can read.

READABILITY

Syntax highlighting

Colors and formatting can make names, strings, comments, and language structure easier to scan.

NAVIGATION

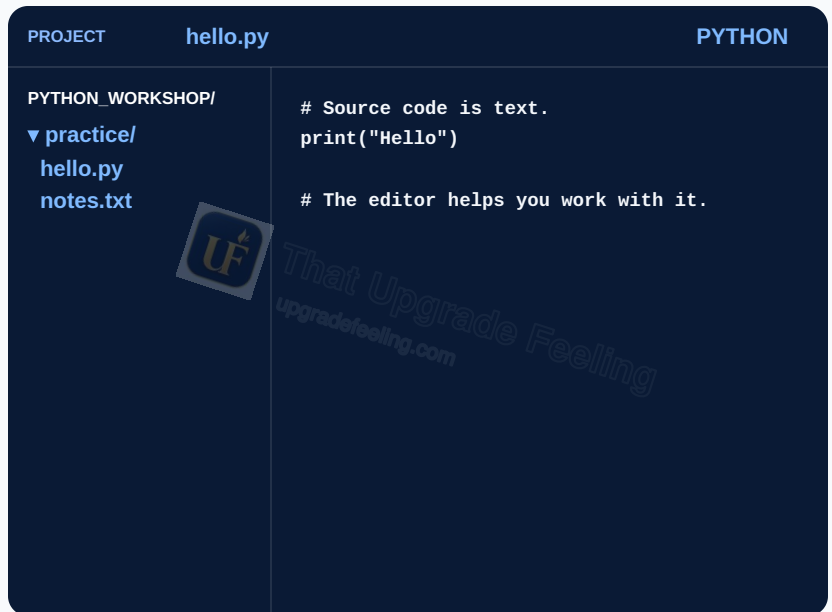
Search and move

Line numbers, search, tabs, file trees, and symbol navigation help you work across a project.

EXTENSIBILITY

Add language support

Many editors can gain linting, completion, debugging, and other features through extensions.



AN EDITOR DOES NOT HAVE TO EXECUTE THE LANGUAGE BY ITSELF

A Run button may call another tool behind the scenes. For Python, the configured Python interpreter is what executes the Python program.

KEY DISTINCTION

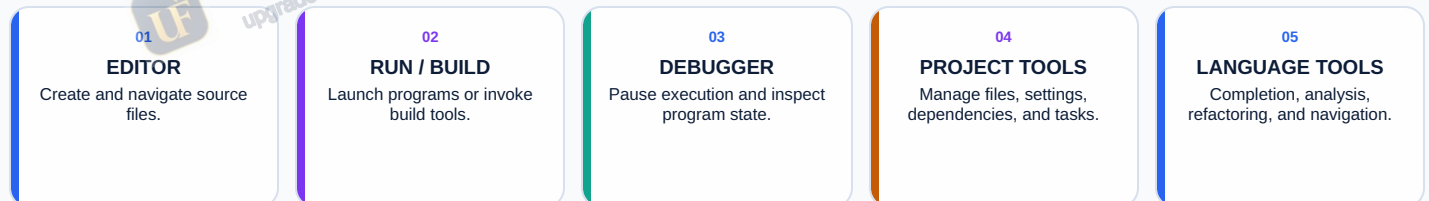
THE EDITOR IS YOUR WORKSPACE FOR SOURCE CODE. IT IS NOT PYTHON, AND IT IS NOT THE PYTHON INTERPRETER.

GET READY - DEVELOPMENT ENVIRONMENT

PAGE 10

What is an IDE?

An IDE - Integrated Development Environment - brings several development tools into one coordinated workspace. The exact feature set varies by product and language.



THE WORD "INTEGRATED" IS THE IMPORTANT PART

An IDE connects tools that could otherwise exist separately. You can edit, run, debug, and navigate without manually switching between unrelated programs for every step.

THE BORDER IS NOT PERFECT

Modern code editors can gain IDE-like features through extensions, while IDEs can be lightweight or modular. Treat "editor" and "IDE" as useful categories, not laws of physics.

BEGINNER MODEL

IDE = EDITOR + INTEGRATED DEVELOPMENT TOOLS

The integration can reduce setup friction, but the underlying concepts - files, interpreter/runtime, terminal commands, debugging, and project structure - still matter.

GET READY - TOOL ROLES

PAGE 11

Editor vs IDE vs interpreter vs terminal

These four terms describe different jobs. They can appear in one application window, but combining them visually does not make them the same thing.

WRITE

Code Editor

Creates and edits source-code files. It may also connect to extensions and external tools.

INTEGRATE

IDE

Combines editing with development tools such as running, debugging, analysis, and project management.

EXECUTE PYTHON

Python Interpreter

Reads and executes Python program instructions according to a Python implementation.

ISSUE COMMANDS

Terminal

A text interface for launching commands and programs, including commands that start the Python interpreter.

EDITOR / IDE

You save hello.py



RUN ACTION

Button or terminal command



PYTHON INTERPRETER

Executes the program



OUTPUT

What the running program produces

QUICK CHECK

You click **Run** in an editor. Which component actually executes the Python program?

The source-code editor itself, because it displays the file.

The configured Python interpreter.

The terminal, even when no terminal command is involved.

Choose an answer.

NEXT: PICK A PRACTICAL STARTING TOOL

PAGE 12 CHOOSES A BEGINNER WORKSHOP SETUP WITHOUT PRETENDING THERE IS ONE UNIVERSAL “BEST” EDITOR.

GET READY - PICK A TOOL

PAGE 12

Which one should you use?

For this workshop, use a setup that is common, capable, and easy to verify. You do not need to identify the perfect editor before you can learn programming.

WORKSHOP DEFAULT

Visual Studio Code

We will use VS Code as the main editor example. With Python installed separately and Python support added to VS Code, it gives you editing, running, terminal, and debugging workflows without hiding the underlying pieces.

WORKSPACE
VS Code

+

LANGUAGE
SUPPORT
Python
extension

+

EXECUTION
Python
interpreter

VALID ALTERNATIVE

PyCharm

A full Python-focused IDE is also a valid choice. The screens and project workflow differ, but the core programming concepts do not.

ALSO POSSIBLE

Another editor or IDE

If it can work reliably with Python files and your chosen interpreter, you can learn with it. Workshop screenshots and steps may not match exactly.

DO NOT INSTALL EVERYTHING

One primary tool is enough

Using several editors at once creates more settings to understand before you have a reason to need them.

WORKSHOP BASELINE

COMPUTER + PYTHON + VISUAL STUDIO CODE. PYCHARM IS A VALID ALTERNATIVE, NOT A REQUIRED SECOND INSTALL.

GET READY - AVOID TOOL PARALYSIS

PAGE 13

Do not waste a week choosing an editor

Beginners can spend more time comparing tools than writing programs. Your first editor is a working environment, not a permanent identity.

RULE 01

Pick for the next task

Choose a tool that can edit .py files, use your Python interpreter, and give you a workable run/debug path.

RULE 02

Learn the concepts underneath

Know what a file, interpreter, terminal, debugger, and project folder are. Then a different interface becomes easier to understand.

RULE 03

Switch when you have a reason

Change tools because a real need appears - not because another editor has a nicer screenshot or a longer feature list.

YOU ARE READY TO STOP COMPARING WHEN...

- 1 You can create and save a Python source file.
- 2 You can point the tool at the Python interpreter you intend to use.
- 3 You can run a file and see its output or error.
- 4 You can open a terminal or equivalent command workflow when needed.

WHAT MATTERS MORE THAN THE BRAND

The transferable skill is understanding the workflow: source file → interpreter/runtime → output → debugging and revision.

Menus, shortcuts, panels, and extension systems change. The underlying responsibilities remain recognizable.

WORKSHOP RULE

**CHOOSE. CONFIGURE.
VERIFY. THEN START
LEARNING.**

NEXT: INSTALL THE LANGUAGE RUNTIME

PAGE 14 STARTS WITH WINDOWS AND USES THE CURRENT OFFICIAL PYTHON INSTALL PATH - NOT AN OLD SCREENSHOT TUTORIAL.

INSTALL & CONNECT - WINDOWS

PAGE 14

Install Python on Windows

Python's official Windows installation path has changed. This guide follows the current Python documentation rather than relying on old installer screenshots.

CURRENT OFFICIAL PATH · CHECKED AUG 2026

Use the Python Install Manager

The CPython team directs Windows users to the Python Install Manager, available from the Microsoft Store or from Python's official downloads site.

WHY OLD VIDEOS MAY LOOK DIFFERENT

The traditional standalone executable installer is being replaced. Current Python documentation centers the Install Manager, so exact buttons in older tutorials may not match your machine.

STEP 01

Start from an official source

Open the Microsoft Store listing for Python Install Manager or the official Python downloads page. Avoid random download mirrors.

STEP 02

Install the manager

Use the Store's Install action, or open the downloaded official installer package and choose Install.

STEP 03

Open a fresh terminal

After the manager is installed, open a new terminal window so Windows can see the newly available commands.

STEP 04

Start with the normal command

The current documentation recommends python as the normal launch command. If no runtime is installed yet, the manager can obtain the current default runtime.

FIRST COMMAND - VERIFICATION COMES ON PAGE 17

```
python
```

If python or py does not work, do not start changing PATH entries at random. Keep the error message; the troubleshooting pages will use it.

VERSION-RESILIENT RULE

Install an actively supported Python 3 runtime from the official Python path. The exact newest version number will change over time; your programming fundamentals do not depend on hard-coding today's release number into this guide.

INSTALL & CONNECT - macOS

PAGE 15

Install Python on macOS

For a straightforward workshop setup, use the current Python installer provided by the CPython release team on python.org. Keep Apple's own system-managed Python components untouched.

OFFICIAL CPYTHON PATH · CHECKED AUG 2026

Use the current macOS installer from python.org

Python's macOS documentation recommends the most recent supported Python version where possible. Current installers are universal builds for supported Apple Silicon and Intel Macs.

DO NOT DELETE APPLE'S PYTHON

Some Macs expose an Apple-controlled `/usr/bin/python3` for developer tools. Python's documentation says not to modify or remove it. A newer python.org installation can coexist with it.

STEP 01

Go to the official macOS downloads page

Choose a current stable Python 3 release from python.org. For this workshop, skip alpha, beta, and release-candidate builds.

STEP 02

Run the signed .pkg installer

Open the downloaded package and use the standard installation unless you have a specific reason to customize it. macOS may ask for an administrator account.

STEP 03

Complete the certificate step

Open the newly installed Python folder in Applications and run `Install Certificates.command`. The official docs include this as the final setup step for the python.org build.

STEP 04

Open a fresh Terminal window

Use `python3` for the command-line interpreter. Page 17 will verify that the command resolves to a working Python 3 installation.

MACOS COMMAND - VERIFICATION COMES ON PAGE 17

`python3`

Do not force python to mean Python 3 just to make another tutorial match. Learn the command your platform actually provides.

WHY NOT HOMEBREW HERE?

Homebrew is a valid third-party distribution path, but it adds another package manager to explain. The workshop baseline uses the official python.org installer so the first setup stays easier to verify.

VERSION-RESILIENT RULE

Use a supported stable Python 3 release. Exact version numbers and installer screenshots will change; the verification workflow on Page 17 is the durable part.

INSTALL & CONNECT - LINUX

PAGE 16

Install Python on Linux

Linux is different from Windows and macOS: Python is already present on many distributions, and the normal beginner path is to use your distribution's package manager rather than replace the system interpreter by hand.

CHECK BEFORE YOU INSTALL

Open a terminal and run `python3 --version`. If it returns a supported Python 3 version, you may already have what this workshop needs.

DO NOT "UPGRADE" SYSTEM PYTHON AT RANDOM

Linux tools can depend on the distribution's Python packages. Do not delete, overwrite, or redirect system Python just to match a tutorial.

UBUNTU

Install the full supported Python environment

```
sudo apt update
```

```
sudo apt install -y python3-full
```

Canonical's current developer guidance uses `python3-full` so the standard interpreter and common environment support are available together.

FEDORA / RHEL FAMILY

Use DNF

```
sudo dnf install python3
```

Use the distribution package rather than downloading an unrelated binary installer.

ARCH LINUX

Use Pacman

```
sudo pacman -S python
```

Arch packages Python under the package name `python`; the installed interpreter is invoked as `Python3`.

LINUX COMMAND - VERIFY FIRST, INSTALL ONLY IF NEEDED

```
python3 --version
```

Different distributions package software differently. If you use another distro, follow its official package documentation instead of copying a command meant for Ubuntu, Fedora, or Arch.

RULE 01

Prefer the distro package

It integrates with the operating system's update and dependency tools.

RULE 02

Use `python3` in the terminal

That command convention avoids old Python 2 assumptions still found in older material.

RULE 03

Source builds are advanced setup

Python can be compiled from source, but that is not the beginner workshop path.

RULE 04

Do not overwrite `python3`

Python's own Unix documentation warns that a careless source install can overwrite or masquerade as the system binary.

INSTALL & CONNECT - VERIFY

PAGE 17

Verify Python is installed

An installer saying "finished" is not enough. Verification means a fresh terminal can find Python and report a Python 3 runtime without you guessing which installation is active.

WINDOWS

Check the normal command

```
python --version
```

A successful result starts with **Python 3** followed by a version number. The exact number can change over time.

macOS

Use the Unix-style command

```
python3 --version
```

The python.org macOS installation is normally invoked from Terminal with python3.

LINUX

Use the distro's Python 3 command

```
python3 --version
```

If Python came with your distro or from its package manager, this is the normal version check.

SUCCESS LOOKS LIKE THIS

EXAMPLE SHAPE - YOUR VERSION MAY DIFFER

```
Python 3.x.y
```

1 · COMMAND FOUND

The shell recognizes the Python command.

2 · PYTHON 3

The reported major version is 3.

3 · REPEATABLE

A fresh terminal returns the same kind of result.

IF THE COMMAND FAILS

- Close the terminal and open a fresh one.
- Confirm you used the command for your operating system.
- Return to the matching install page and confirm the installation completed.
- Keep the exact error message for the troubleshooting section.
- Do not edit PATH or replace system Python at random.

WHAT YOU HAVE PROVED

Your terminal can reach a Python 3 interpreter

That is the first real setup checkpoint. You have not configured VS Code yet - editor integration comes next.

DO NOT CONFUSE THE PIECES

A VS Code Python extension does not install Python itself, and installing Python does not automatically tell every editor which interpreter to use.

NEXT: INSTALL THE WORKSHOP EDITOR

PAGE 18 INSTALLS VISUAL STUDIO CODE. PAGE 19 ADDS PYTHON SUPPORT. PAGE 20 CONNECTS VS CODE TO THE INTERPRETER YOU JUST VERIFIED.

INSTALL & CONNECT - EDITOR

PAGE 18

Install Visual Studio Code

You already have Python. Now install the workshop editor. Visual Studio Code is a separate application: it gives you the workspace where you write and organize code, but it does not replace the Python interpreter.

WORKSHOP BASELINE · CHECKED AUG 2026

Download VS Code from the official Visual Studio Code site

Use the stable release for your operating system. Avoid third-party download mirrors and skip Insiders builds for this beginner setup.

ONE EDITOR · THREE INSTALL PATHS

The application is the same VS Code, but Windows, macOS, and Linux package it differently. Follow only the card for your operating system.

WINDOWS

Use the User Setup

Microsoft recommends User Setup for most people. It does not require administrator permissions and supports the smoothest update experience.

Run `VSCodeUserSetup-{version}.exe` and complete the standard installer.

macOS

Open the .dmg

Download the macOS build, open the disk image, then drag Visual Studio Code.app into the Applications folder.

Launch it from Applications when installation is complete.

LINUX

Match your distribution

VS Code is officially available as Debian, RPM, and Snap packages. Use the package type that fits your distribution.

On Debian/Ubuntu, the official .deb package is the straightforward workshop path.

01 · DOWNLOAD

Use the stable official build

Choose the package for your OS and hardware.

02 · INSTALL

Keep the default setup

No special themes, profiles, or extensions are needed yet.

03 · OPEN

Launch VS Code once

Confirm the application opens to its normal editor window.

04 · STOP HERE

Do not install random add-ons

Page 19 adds only the Python support we actually need.

OPTIONAL COMMAND-LINE CONVENIENCE

code .

Windows User Setup adds VS Code to PATH; on macOS, the official setup docs provide a Command Palette action to install the code command. This shortcut is useful later, but it is not required to continue.

CHECKPOINT

VS CODE OPENS SUCCESSFULLY. PYTHON IS STILL A SEPARATE INSTALLATION. NEXT, WE ADD THE BRIDGE BETWEEN THEM.

INSTALL & CONNECT - PYTHON SUPPORT

PAGE 19

Add Python support to VS Code

A fresh VS Code install is a general-purpose editor. Add Microsoft's Python extension so VS Code understands Python files and can provide Python-specific editing, running, debugging, and environment integration.

INSTALL THE OFFICIAL PYTHON EXTENSION

- Open the **Extensions** view from the Activity Bar.
- Or use **Ctrl+Shift+X** on Windows/Linux; on macOS use the Extensions shortcut shown by VS Code.
- Search for **Python**.
- Choose **Python** published by **Microsoft**.
- Select **Install**.

Extension ID: `ms-python.python`

VERIFY THE PUBLISHER

MS

Publisher: Microsoft

Check the publisher and extension identifier instead of installing the first similarly named result.

VS Code's extension model allows many publishers to ship tools. For the workshop baseline, use the official Microsoft Python extension.

WHAT THE EXTENSION ADDS

Python-aware features inside the editor

It connects VS Code's interface to Python workflows such as IntelliSense, running, debugging, and interpreter selection.

WHAT IT DOES NOT ADD

The Python extension does **not** contain the Python interpreter. Pages 14–17 installed and verified Python separately for exactly this reason.

EDITOR

VS Code

The workspace where you open folders and edit files.

EXTENSION

Python support

Adds Python-specific commands and editor integration.

INTERPRETER

Python runtime

The executable that actually runs your program.

CONNECTION

Select interpreter

Page 20 tells VS Code which Python runtime to use.

AFTER INSTALL

The Install button changes to the extension's management controls. You do not need to install a large bundle of unrelated extensions. Keep the setup small until you understand what each tool contributes.

NEXT CONNECTION

VS CODE NOW UNDERSTANDS PYTHON WORKFLOWS. PAGE 20 SELECTS THE INTERPRETER THAT WILL ACTUALLY EXECUTE YOUR CODE.

INSTALL & CONNECT - INTERPRETER

PAGE 20

Select the Python interpreter

Your computer can contain more than one Python installation. Interpreter selection tells VS Code which environment should run your files and power Python features for the current workspace.

STEP 01**Open a Python context**

Open a folder or a .py file in VS Code so Python tools have a workspace to work with.

STEP 02**Open the Command Palette**

Use Ctrl+Shift+P on Windows/Linux or Cmd+Shift+P on macOS.

STEP 03**Run the selection command**

Type and choose Python: Select Interpreter.

STEP 04**Choose your verified Python**

Select the supported Python 3 installation you set up and verified on Pages 14–17.

HOW TO CHOOSE WHEN SEVERAL APPEAR

- Prefer the Python 3 installation you intentionally installed for the workshop.
- Check the displayed version and path rather than guessing from the first item.
- Do not deliberately select an old, broken, or unrelated environment.
- Later, project-specific virtual environments may become the better choice.

THE PATH IS A CLUE

VS Code discovers Python from places such as PATH, known install locations, system directories, and workspace environments. A path helps distinguish one interpreter from another.

Windows example shape: ...\\Python...\\python.exe

macOS/Linux example shape: .../bin/python3

CONFIRM THE ACTIVE ENVIRONMENT

Look at the Python environment control in VS Code's Status Bar. You can also select that control to switch environments. Current VS Code environment tooling automatically discovers common Python installations.

WHY THIS CHOICE MATTERS

The selected environment is used for running code, debugging, and language features such as IntelliSense. Selecting the wrong interpreter can make installed packages appear to be "missing."

MENTAL MODEL - KEEP THESE ROLES SEPARATE

VS Code → Python extension → selected interpreter → program output

The editor does not execute Python by itself. The selected interpreter is the runtime behind the Run command.

SETUP MILESTONE

COMPUTER + PYTHON + VS CODE + PYTHON EXTENSION + SELECTED INTERPRETER ARE NOW CONNECTED AS ONE WORKING TOOLCHAIN.

FILES & WORKFLOW - WORKSPACE

PAGE 21

Open your workshop folder in VS Code

Work from a folder, not a loose file. Opening the workshop folder gives VS Code a workspace context for the Explorer, Python environment, terminal location, and the files you create next.

OPEN THE FOLDER

1

Create it once if needed

Make a folder somewhere you can find again. Use Python_Workshop for this setup.

2

Choose File > Open Folder...

In VS Code, select the folder itself - not an individual file inside it.

3

Confirm the Explorer

The folder name should appear at the top of the Explorer view on the left.

WORKSPACE CHECK

If VS Code shows a Workspace Trust prompt, read it. A folder you created yourself for this workshop is a normal trusted workspace; do not automatically trust unknown downloaded projects.

EXPLORER - EXPECTED SHAPE

PYTHON_WORKSHOP/
(empty for now)

An empty Explorer is fine. Page 22 creates the first source file inside this folder.

FOLDER

Known location

You know where the project lives on disk.

EXPLORER

Folder is visible

VS Code shows the workspace contents.

CONTEXT

Tools share one root

Files, terminal, and Python tooling can work from the same folder.

OPTIONAL TERMINAL SHORTCUT - ONLY IF THE `code` COMMAND IS AVAILABLE

```
cd path/to/Python_Workshop code .
```

The `.` means the current folder. The menu route **File > Open Folder...** is enough for the workshop and works without this shortcut.

CHECKPOINT

THE EXPLORER SHOWS PYTHON_WORKSHOP AS YOUR OPEN FOLDER. NOW CREATE A REAL PYTHON SOURCE FILE INSIDE IT.

FILES & WORKFLOW - FIRST SOURCE FILE

PAGE 22

Create your first Python file

A Python program can begin as a plain-text source file ending in `.py`. Create it inside the open workshop folder so the file, selected interpreter, and run tools all share the same workspace.

CREATE THE FILE FROM EXPLORER

- In the Explorer, select **New File**.
- Name the file exactly `hello_workshop.py`.
- Press Enter to create and open it.
- Type the one-line program shown here.
- Save the file with `Ctrl+S` on Windows/Linux or `Cmd+S` on macOS.

`hello_workshop.py`

TYPE THIS YOURSELF

`hello_workshop.py`

PYTHON

```
print("Python Workshop is ready.")
```

The goal is not the complexity of the code. The goal is to prove that you can create, save, and recognize a real Python source file before running it.

EXTENSION

`.py` identifies Python source

Do not accidentally create `hello_workshop.py.txt`.

PLAIN TEXT

The file stores code characters

It is not a Word document or rich-text file.

SAVED STATE

Save before the first run

The file on disk should contain the line you typed.

WHAT VS CODE SHOULD NOTICE

The editor now has Python context

The `.py` extension lets VS Code and the installed Python tooling treat this as Python source code.

DO NOT CONFUSE RECOGNITION WITH EXECUTION

Syntax coloring or a Python label means the editor recognizes the file type. The selected Python interpreter still performs the actual execution.

CHECKPOINT

HELLO_WORKSHOP.PY EXISTS INSIDE PYTHON_WORKSHOP, IS SAVED, AND CONTAINS ONE PRINT() STATEMENT. NEXT, RUN IT.

FILES & WORKFLOW - FIRST RUN

PAGE 23

Run Python from VS Code

Now complete the local development loop. VS Code can send the active `.py` file to the selected Python interpreter and show the process and output in its integrated terminal.

01 · OPEN

Keep `hello_workshop.py` active

The editor tab should show the file you want to run.

02 · SAVE

Save your latest text

Use the saved file as the source of truth for this first run.

03 · RUN

Select Run Python File in Terminal

Use the play button in the top-right of the Python editor.

04 · INSPECT

Read the terminal output

Confirm the command ran and the expected message appeared.

RUN CONTROL

`hello_workshop.py`

▶ Run Python File

Microsoft's Python extension provides **Run Python File in Terminal**. It opens the terminal panel and runs the active file using the selected Python environment.

EXPECTED TERMINAL RESULT

COMMAND SHAPE MAY DIFFER BY OS / ENVIRONMENT

```
... hello_workshop.py
Python Workshop is ready.
```

The important evidence is the final output line - not whether your command begins with `python`, `python3`, or a full interpreter path.

No Run Python button?

Confirm the file ends in `.py` and the Microsoft Python extension is installed.

Wrong Python environment?

Use Python: Select Interpreter and choose the verified environment from Page 20.

No expected output?

Save the file, check the exact `print()` line, run again, and read any terminal error instead of guessing.

YOUR FIRST COMPLETE LOCAL LOOP

folder -> `.py` file -> selected interpreter -> terminal -> output

You have now moved from installing tools to using them as a connected workflow. Page 24 slows down and explains the terminal you just saw.

NEXT: MEET THE TERMINAL

YOU HAVE USED THE TERMINAL THROUGH VS CODE. NEXT, LEARN WHAT THE TERMINAL IS AND WHY IT MATTERS.

FILES & WORKFLOW - TERMINAL BASICS

PAGE 24

Meet the terminal

The terminal is a text-based place to enter commands and read their output. In VS Code, the integrated terminal keeps that command-line workflow beside your source files instead of forcing you to switch to a separate app.

TERMINAL**The panel you interact with**

It displays text input, command output, errors, and prompts.

SHELL**The command interpreter**

PowerShell, Bash, and Zsh are shells that read commands and launch programs.

PROMPT**The shell is ready**

The prompt marks where your next command can be entered. Its exact appearance varies.

WORKING DIRECTORY**Your current folder**

Relative file names are interpreted from the directory the shell is currently using.

OPEN THE INTEGRATED TERMINAL

- Keep Python_Workshop open as your workspace.
- Choose **View > Terminal** or **Terminal > New Terminal**.
- VS Code normally starts the terminal at the workspace folder root.
- Look at the prompt before typing anything.

PROMPT EXAMPLES - YOURS MAY LOOK DIFFERENT

```
PS C:\...\Python_Workshop>
...\Python_Workshop $
```

TERMINAL IS NOT THE SAME AS SHELL

VS Code provides the terminal interface. Inside it, your operating system supplies a shell. The shell interprets the command you type and can launch programs such as Python.

YOU
type

TERMINAL
shows

SHELL
interprets

PROGRAM
runs

IMPORTANT

The terminal itself does not execute Python code. The shell launches a Python interpreter, and that interpreter executes the `.py` file.

NEXT: RUN THE FILE YOURSELF

PAGE 25 USES THE TERMINAL DIRECTLY SO YOU CAN SEE THE COMMAND THAT LAUNCHES YOUR PYTHON FILE.

FILES & WORKFLOW - MANUAL RUN

PAGE 25

Run Python from the terminal

Now run the same saved file without the editor's Run button. The command makes the execution path explicit: ask a Python interpreter to open and execute `hello_workshop.py`.

START FROM THE WORKSHOP FOLDER

- Open the integrated terminal.
- Confirm the prompt is inside `Python_Workshop`.
- Use the Python command you verified on Page 17.
- Type the file name after the command and press Enter.

WINDOWS

```
python hello_workshop.py
```

macOS / LINUX

```
python3 hello_workshop.py
```

EXPECTED RESULT

THE COMMAND ECHO / PROMPT VARIES - THE OUTPUT SHOULD MATCH

```
python hello_workshop.py
Python Workshop is ready.
```

On macOS or Linux, the first line will normally begin with `python3`. If your verified installation or active environment uses another executable path, use that instead.

WHY THE CURRENT FOLDER MATTERS

`hello_workshop.py` is a relative file name. Python looks for it from the shell's working directory unless you provide a different path.

Command not found?

Return to Page 17 and verify the Python command your operating system recognizes.

Can't open file?

Check the working directory and exact file name. The script must exist where the command expects it.

Output is different?

Open the saved file, inspect the `print()` text, save, and run the command again.

MANUAL EXECUTION MODEL

```
shell prompt -> python / python3 -> hello_workshop.py -> output
```

You are not using a different Python language here. You are simply launching the interpreter yourself instead of asking the editor to construct the run command for you.

NEXT: TWO ROUTES, SAME CORE

PAGE 26 COMPARES THE RUN BUTTON WITH A MANUAL TERMINAL COMMAND SO YOU KNOW WHEN EACH WORKFLOW IS USEFUL.

FILES & WORKFLOW - RUN WORKFLOWS

PAGE 26

Run button vs terminal

Both routes can execute the same Python file. The difference is how much of the command is prepared for you and how much control you want over what is launched.

RUN PYTHON FILE

Fast path from the editor

- Runs the active Python file.
- Uses the Python environment selected in VS Code.
- Opens or uses a terminal and constructs the run command for you.
- Good for the normal edit - run - inspect loop.

VS

MANUAL TERMINAL COMMAND

Explicit command-line path

- You choose the command and file explicitly.
- You can add arguments, run tools, or launch a different file.
- You see exactly what the shell is being asked to execute.
- Useful for learning, automation, and troubleshooting.

EDITOR
source file

RUN ROUTE
button or command

PYTHON
interpreter

PROGRAM
executes

TERMINAL
shows output

Use the Run button when...

You are repeatedly running the active file and want a low-friction development loop.

Use the terminal when...

You need an explicit command, arguments, another command-line tool, or a clearer troubleshooting path.

Do not turn this into a status contest

The terminal is not automatically more professional, and the Run button is not fake. Competent developers use both.

DO THIS NOW

Run the same file both ways

Use **Run Python File in Terminal**, then type the manual command from Page 25. Both runs should print `Python Workshop is ready.`

WHAT YOU JUST PROVED

The editor control and the terminal command are two interfaces into the same underlying execution idea: a Python interpreter runs your saved source file.

NEXT: DEBUGGER

RUNNING TELLS YOU WHAT HAPPENED. PAGE 27 INTRODUCES A TOOL THAT HELPS YOU INSPECT HOW THE PROGRAM REACHED THAT RESULT.

FILES & WORKFLOW - DEBUGGING

PAGE 27

What is a debugger?

Running a program tells you what happened. A debugger helps you inspect *how* it happened by controlling execution, pausing at selected lines, and showing the program's current state.

NORMAL RUN

Execute until the program finishes or fails

You launch the file, read its output or error, then decide what to change.

VS

DEBUG SESSION

Pause and inspect the program while it runs

You can stop at a breakpoint, inspect variables, step through statements, and continue execution.



That Upgraded Feeling
upgradefeeling.com



That Upgraded Feeling
upgradefeeling.com

01
Set breakpoint02
Start debugger03
Pause on line04
Inspect state05
Step / continue

hello_debug.py

BREAKPOINT ON LINE 03

```
price = 12
quantity = 3
● total = price * quantity
print(total)
```

VARIABLES

PAUSED

price	12
quantity	3
total	not assigned yet

Breakpoint

A marker that asks the debugger to pause when execution reaches that line.

Python Debugger extension

VS Code's Python debugging support is provided through the Python Debugger extension, installed with the Python extension.

Debugger does not repair code

It exposes evidence. You still interpret the program state and decide what the bug is.

TRY IT LATER - NOT REQUIRED YET

Breakpoint + F5 / Run and Debug

For a simple Python file, set a breakpoint in the editor gutter and start a debug session. Custom `launch.json` configuration becomes useful when the run needs persistent options such as arguments or a special launch mode.

WHY LEARN THIS EARLY?

Print statements are useful, but a debugger gives you a structured way to observe state and execution order when a program becomes harder to reason about.

NEXT: TWO DIFFERENT KINDS OF ADD-ONS

PAGE 28 SEPARATES VS CODE EXTENSIONS FROM PYTHON PACKAGES SO YOU KNOW WHAT EACH INSTALLATION ACTUALLY CHANGES.



That Upgraded Feeling
upgradefeeling.com

FILES & WORKFLOW - ADD-ONS

PAGE 28

Extensions vs Python packages

Both can add capability, but they are installed into different systems. A VS Code extension changes or extends the editor. A Python package is installed into a Python environment for Python programs and tools to use.

VS CODE EXTENSION

Adds capability to the editor

- Installed from the VS Code Marketplace.
- Can add language support, debuggers, themes, formatters, and other editor tools.
- Example: Python by Microsoft; Python Debugger.

VS

PYTHON PACKAGE

Adds installable Python software to an environment

- Usually installed with a Python package manager such as pip.
- Lives in a Python environment, not in the VS Code Marketplace.
- May provide importable modules, command-line tools, or both.

EDITOR LANE

VS Code -> Extensions view -> Marketplace -> extension

The extension integrates with the editor and VS Code APIs.

PYTHON ENVIRONMENT LANE

project environment -> pip -> distribution package

The installed package belongs to that Python environment. Later, the workshop will use project virtual environments for third-party packages.

QUESTION

Want syntax support or a debugger?

ANSWER

Look for a VS Code extension.

QUESTION

Want a library your Python code uses?

ANSWER

Look for a Python package.

Do not install every package globally

The Python Packaging User Guide recommends using virtual environments for third-party packages so project dependencies stay isolated. We will cover that workflow later.

Package has two meanings in Python discussions

Packaging documentation distinguishes a downloadable *distribution package* from an *import package* used in source code. For now, remember the installation destination: the Python environment.

NEXT: WHERE DOES THE WORK ACTUALLY RUN?

PAGE 29 COMPARES A LOCAL DEVELOPMENT WORKSPACE WITH BROWSER-BASED CODING WITHOUT PRETENDING THAT EVERY BROWSER TOOL HAS THE SAME CAPABILITIES.

LEARN WELL & START - WORKSPACE MODEL

PAGE 29

Local vs browser-based coding

The editor you see is only part of the environment. The important question is where the files, Python runtime, terminal, extensions, and compute actually live.

LOCAL WORKSPACE

Your computer owns the development environment

- Files live on your machine.
- Your installed Python interpreter runs the program.
- You have normal access to local folders, terminal, and desktop extensions.
- Many exercises can keep working without a constant internet connection once tools are installed.

VS

BROWSER-BASED WORKSPACE

The browser is the front end; execution capability varies

- Some tools focus on editing and repository browsing.
- Some run code inside the browser using technologies such as WebAssembly.
- Some connect the browser to remote compute or a cloud development environment.
- Terminal, debugger, extension, and file-system capability depends on the service.

EDITOR IN THE BROWSER

Lightweight / sandboxed

vscode.dev can browse and edit code with zero installation, but the web sandbox constrains parts of the full desktop workflow.

IN-BROWSER RUNTIME

Experimental Python in the web

VS Code has experimental Python-in-the-Web support based on WebAssembly. It requires a separate experimental extension and specific prerequisites, so it is not equivalent to the normal desktop Python workflow.

REMOTE COMPUTE

Browser UI, machine elsewhere

Cloud or remote environments can provide a fuller runtime and terminal while you interact through a browser.

WHY THIS WORKSHOP USES LOCAL DESKTOP FIRST

You are learning the normal relationship between folders, interpreter, terminal, editor, and debugger. A local computer makes those boundaries easier to see and gives you a baseline that is not tied to one web service.

THIS IS NOT "LOCAL GOOD, BROWSER BAD"

Browser coding is useful for quick access, constrained devices, collaboration, remote machines, and temporary environments. Choose it when its execution model matches the task.

IMPORTANT NUANCE

"Runs in a browser" does not tell you where your Python code runs.

The runtime may be inside the browser sandbox, on a remote machine, or absent entirely. In VS Code for the Web, Python execution through WebAssembly is experimental and has important limitations. Always check where execution happens and what file, terminal, debugger, extension, package, network, and threading capabilities are actually available.

NEXT: WHY A COMPUTER IS THE WORKSHOP BASELINE

PAGE 30 TURNS THIS COMPARISON INTO A PRACTICAL RECOMMENDATION FOR HOW TO LEARN WITHOUT FIGHTING YOUR TOOLS.

LEARN WELL & START - DEVICE BASELINE

PAGE 30

Why we recommend a computer

Not because a phone or tablet is incapable of code. A computer is the workshop baseline because it exposes the normal development workflow with fewer artificial constraints between your files, editor, interpreter, terminal, and debugger.

THE BASELINE

Desktop or laptop - Windows, macOS, or Linux

The important part is not the brand. You need a general-purpose computer where you can manage project folders, install tools, open a terminal, and run your own Python interpreter.

THIS IS A LEARNING RECOMMENDATION, NOT A RULE

If a phone, tablet, browser environment, or remote machine is what you have, you can still learn. Just expect the exact file, terminal, extension, and runtime workflow to differ.

FILES

See the real project structure

Create folders and files, rename them, move them, and understand where your program actually lives.

TOOLS

Install the normal stack

Use Python, VS Code, extensions, and later project-specific tooling without depending on one browser service.

TERMINAL

Run commands directly

Learn working directories, commands, interpreter invocation, and output in the same environment as your files.

DEBUGGING

Inspect a running program

Use breakpoints, stepping, and variable inspection as programs become less obvious.

MULTI-FILE WORK

Grow beyond one snippet

Keep source files, notes, exercises, and later environments together in one visible workspace.

TRANSFER

Learn a model that travels

The local folder + editor + terminal + interpreter model remains useful even when you later move to cloud or remote development.

PHONE / TABLET = GREAT COMPANION

Read lessons, review code, make small edits, test quick ideas, or use a browser environment when a computer is unavailable.

DO NOT DELAY LEARNING FOR PERFECT HARDWARE

Use a reasonable machine you already have. For this foundations workshop, the workflow matters far more than chasing premium specifications.

Choose the least restrictive environment that lets you practice the full loop: create -> run -> inspect -> change -> run again.

That is why the workshop recommends a computer as the primary workspace while still treating mobile and browser tools as legitimate alternatives for the right situation.

NEXT: AI AS A COACH, NOT AN ESCAPE HATCH

PAGE 31 SHOWS HOW TO GET HELP FROM AI WITHOUT OUTSOURCING THE REASONING PRACTICE YOU ARE HERE TO BUILD.

LEARN WELL & START - AI-ASSISTED LEARNING

PAGE 31

Use AI without giving away the thinking

AI can explain, question, compare, and review. It can also erase the exact practice you need if every difficulty is immediately converted into a finished solution. During foundations, use assistance in a way that keeps you responsible for the model in your head.

01

TRY

Make a real attempt before asking for the answer.

02

OBSERVE

Run it. Capture the output, error, or behavior.

03

ASK

Request a hint, explanation, question, or comparison.

04

VERIFY

Test every claim against your program and tools.

05

REBUILD

Close the answer and reproduce the idea yourself.

GOOD USES WHILE LEARNING

- Explain an error message in plain language.
- Ask you questions instead of revealing the solution.
- Compare two approaches you already attempted.
- Generate test cases after you write the logic.
- Review your explanation and point out gaps.

WATCH FOR THESE SHORTCUTS

- Requesting the complete exercise before trying.
- Copying code you cannot explain line by line.
- Accepting a confident answer without running it.
- Replacing debugging with repeated prompt-and-paste cycles.
- Letting AI choose every name, step, and decision for you.

COACH ME

"Do not solve it yet. Ask me one question at a time to help me find the mistake."

EXPLAIN EVIDENCE

"Explain this traceback line by line. Tell me what to inspect, but do not rewrite my program."

WEAK FOR FOUNDATIONS

"Write the whole exercise for me so I can paste it and move on."

The goal is not to avoid AI-generated code forever. The goal is to know when generation helps the work and when it steals the repetition that builds your skill.

In real projects, AI can accelerate implementation. In a foundations exercise, the learning objective may be the reasoning itself. Protect that objective first, then use the tool deliberately.

NEXT: PROVE YOUR SETUP

PAGE 32 TURNS THE ENTIRE PRE-ISSUE SO FAR INTO A CHECKLIST YOU CAN VERIFY ON YOUR OWN MACHINE.

LEARN WELL & START - VERIFICATION

PAGE 32

Setup checklist

A setup is finished when you can prove the pieces work together. Check the result on your own machine; do not mark an item complete because an installer said “done.”

FOUNDATION

Machine + tools

- ☐ **I have a primary computer or a clearly chosen alternative.**
I know which environment will hold my files and run Python.
- ☐ **Python 3 is installed.**
The terminal returns a version instead of “command not found.”
- ☐ **VS Code is installed.**
I can open the editor normally.
- ☐ **Python support is installed in VS Code.**
I can see the Microsoft Python extension enabled.

CONNECTION

Editor + interpreter

- ☐ **My workshop folder is open.**
The Explorer shows the folder, not just one loose file.
- ☐ **I selected the intended Python interpreter.**
I can identify the environment VS Code will use.
- ☐ **I created and saved a .py file.**
The file lives inside the workshop folder.
- ☐ **Run Python File in Terminal works.**
The expected output appears in the integrated terminal.

UNDERSTANDING

Know what is doing what

- ☐ **I can run the same file from the terminal.**
I understand the command depends on OS/environment.
- ☐ **I can distinguish editor, terminal, interpreter, and debugger.**
They cooperate, but they are not the same tool.
- ☐ **I know extensions are not Python packages.**
They install into different systems for different purposes.
- ☐ **I have an AI learning rule.**
I will attempt, verify, and explain - not only paste.

MINIMUM PROOF FROM THE TERMINAL

```
Windows:
python --version
python hello_workshop.py

macOS / Linux:
python3 --version
python3 hello_workshop.py
```

THE OUTPUT IS YOUR EVIDENCE

You should see a Python 3 version and the message from the file you created. The exact version number and interpreter path can differ; what matters is that you know which Python is running and that it can execute your saved file.

CHECKPOINT

If these items are true, your environment is ready for real programming practice.

You do not need a perfect setup. You need a setup you understand well enough to create a file, run it, inspect the result, and recover when something is wrong.

NEXT: TROUBLESHOOTING BEFORE YOU PANIC

PAGE 33 BUILDS A SMALL DIAGNOSTIC ROUTINE FOR THE MOST COMMON SETUP FAILURES BEFORE YOU CHANGE RANDOM SETTINGS.

LEARN WELL & START - TROUBLESHOOTING

PAGE 33

Troubleshooting before you panic

Setup failures feel random when every layer is changed at once. Treat the problem like a small investigation: preserve the exact symptom, identify the layer that failed, prove one assumption, change one thing, then repeat the same test.

01

READ

Keep the exact message or behavior. Do not replace it with "it does not work."

02

LOCATE

Is the failure in the file, folder, terminal, interpreter, extension, or command?

03

PROVE

Run the smallest check that can confirm or reject one assumption.

04

CHANGE ONE THING

Avoid five simultaneous fixes. You need to know what changed the result.

05

RE-RUN

Use the same proof again. A different test can hide whether the fix worked.

SYMPTOM 01**python or python3 is not found**

Reopen the terminal, repeat the version check from Page 17, and confirm Python is actually installed before editing PATH or reinstalling.

SYMPTOM 02**VS Code uses the wrong Python**

Run Python: Select Interpreter, choose the intended environment, then run the file again and verify the selected interpreter.

SYMPTOM 03**The file cannot be found**

Check the working directory and exact filename. List the current folder before adding more path text.

SYMPTOM 04**The Run button or Python tools are missing**

Make sure a saved .py file is active and the Microsoft Python extension is enabled.

SYMPTOM 05**A package is missing in one place but installed in another**

Packages belong to Python environments. First confirm which interpreter is actually running.

SYMPTOM 06**The integrated terminal itself will not launch**

Open a fresh terminal and check the selected shell/profile. Troubleshoot terminal launch separately from Python.

RETURN TO A KNOWN PROOF

Windows:

```
python --version
python hello_workshop.py
```

macOS / Linux:

```
python3 --version
python3 hello_workshop.py
```

ASK FOR HELP WITH EVIDENCE

Share the exact command, exact error, operating system, selected interpreter, and what you expected. The smallest reproducible command is more useful than "Python is broken."

THE TROUBLESHOOTING RULE**Do not repair what you have not identified.**

Random PATH edits, repeated reinstalls, and copied commands can create a second problem. Diagnose the smallest failing layer, make one controlled change, and verify it.

NEXT: YOU ARE READY

PAGE 34 CLOSES PRE-ISSUE 00C AND HANDS THE VERIFIED WORKSPACE TO ISSUE 01.

LEARN WELL & START - HANDOFF

PAGE 34

You are ready - continue to Issue 01

The goal of this pre-issue was never to build the fanciest setup. It was to remove uncertainty between you and the first real program. You now have a workspace you can explain, verify, and recover when something goes wrong.

PRE-ISSUE 00C COMPLETE

Your setup is no longer a black box.

You know where the file lives, which Python runs it, how VS Code connects to that interpreter, where output appears, what the terminal is doing, and where the debugger fits.

READY DOES NOT MEAN PERFECT

You will still meet errors, new tools, packages, environments, and unfamiliar commands. The upgrade is that you now have a baseline and a diagnostic method instead of guessing.

FILES

You can create and locate a Python file

Your project is a real folder you can see and manage.

PYTHON

You can verify the interpreter

A version check proves Python is available before you blame the editor.

VS CODE

You can connect editor and interpreter

The Python extension and selected interpreter have distinct roles.

RUN

You can execute the same file two ways

Use the VS Code Run action or issue the command yourself in the terminal.

INSPECT

You know what the debugger is for

Breakpoints and stepping help when output alone does not explain behavior.

RECOVER

You have a troubleshooting routine

Preserve the symptom, isolate the layer, change one thing, and verify again.

NEXT ISSUE

ISSUE 01 - START CODING WITH PYTHON

The environment is ready. The next job is to use it repeatedly until writing, saving, running, reading output, and correcting mistakes become normal.

CARRY THIS LOOP INTO EVERY LESSON

01 - WRITE

Put the idea into a real .py file.

02 - RUN

Execute it with the selected Python interpreter.

03 - INSPECT

Read the output or error before changing the code.

THE HANDOFF

STOP PREPARING. START PROGRAMMING.

Keep this pre-issue as your setup reference. When the environment works, return your attention to the program in front of you.

CONTINUE

PYTHON WORKSHOP - ISSUE 01 - START CODING WITH PYTHON.