Installing Python and Setting Up Your Development Environment

From a Blank Computer to a Working Python Workspace

Category: Python Development

haas.dev

https://dev-roast-app.vercel.app

---

1. Before You Write Python Code

In the previous guide, you learned what Python is, why it exists, where it is used, and how Python code fits into a real software system.

Now we move from **understanding Python** to actually preparing your computer to use it.

A beginner often thinks:

> "I just need to install Python and start coding."

In reality, a useful Python development environment contains several pieces working together:

```
Your Computer
     ↓
Python Installation
     ↓
Python Interpreter
     ↓
Code Editor
     ↓
Terminal
     ↓
Python Project
     ↓
Your Program
```

Understanding these pieces now will prevent many common beginner problems later.

---

2. What You Will Build in This Guide

By the end of this guide, you should have a working Python development environment and understand:

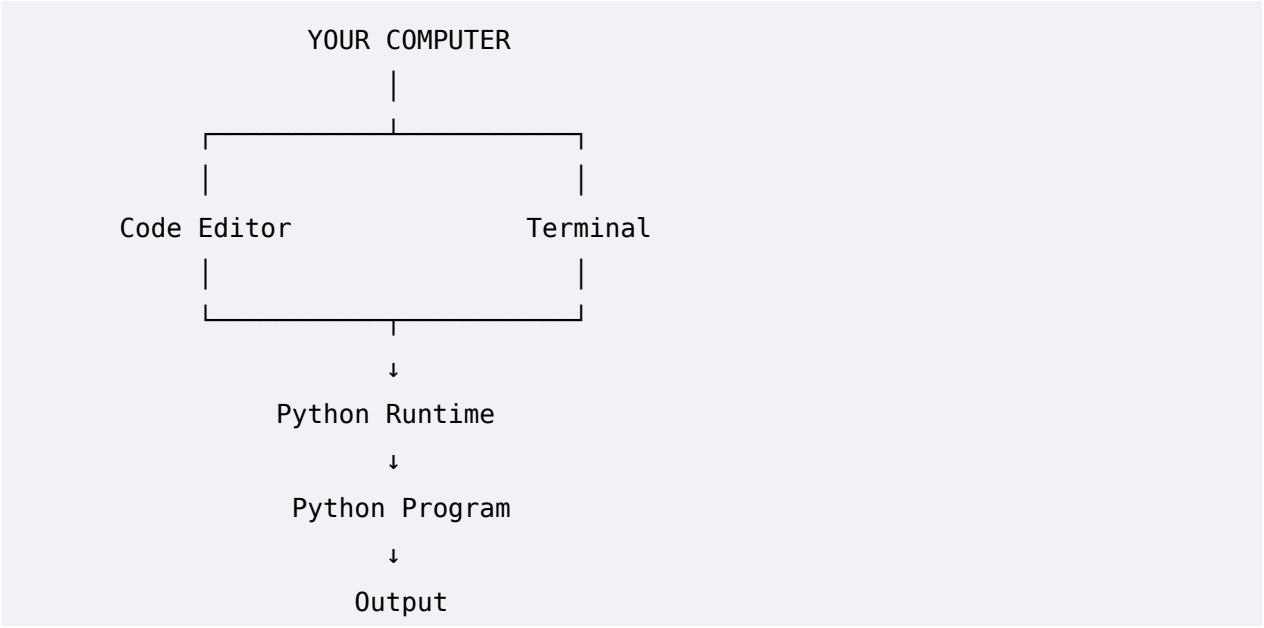
- What Python installation actually provides

- How to download Python safely
- How to install Python on Windows
- How to verify the installation
- What `python` and `python3` mean
- What the Python interpreter is
- How to use the Python interactive shell
- How to install and configure VS Code
- How to install the Python extension
- How to create a Python project
- How to create and run a `.py` file
- How the terminal interacts with Python
- What a PATH variable does
- Why virtual environments exist
- How to recognize common installation problems
- How to verify that your environment is ready

---

3. The Development Environment Mental Model

Before installing anything, understand the relationship between the tools.



Each component has a different responsibility.

Python

Provides the runtime that executes Python programs.

Code Editor

Allows you to write and manage your code.

Terminal

Allows you to execute commands and interact with your development environment.

Project Folder

Keeps your application's files organized.

These tools work together, but they are not the same thing.

---

4. What Does "Installing Python" Actually Mean?

When you install Python, you are not installing a programming language as a physical object.

You are installing software that provides an implementation of Python and the tools needed to run Python programs.

A typical installation provides things such as:

```
Python Runtime
      +
Standard Library
      +
Package Management Tools
      +
Command Line Tools
```

The exact contents can vary depending on the distribution and operating system.

After installation, your computer should be able to execute a command such as:

```
python --version
```

and return the installed Python version.

---

5. Which Python Version Should You Install?

Python has multiple versions.

For a new project, you generally want a **current supported stable Python release**, rather than an obsolete version.

Do not randomly install an old tutorial's version just because the tutorial was published years ago.

Why?

Because older versions can have:

- missing features
- outdated security fixes

- incompatible packages
- unsupported dependencies
- different behavior from current Python

At the same time, the newest release may occasionally have compatibility limitations with a particular third-party package.

Therefore, professional development considers:

```
Project Requirements
        ↓
Supported Python Version
        ↓
Compatible Dependencies
        ↓
Stable Environment
```

For learning Python, use a current supported stable release unless your course or project specifically requires another version.

---

6. Download Python From the Official Source

Python should be downloaded from the official Python website.

Python.org

Avoid downloading Python installers from random websites.

A programming environment becomes part of your development infrastructure, so its source matters.

When downloading software, prefer:

```
Official Website
      ↓
Official Installer
      ↓
Verified Installation
```

rather than:

```
Random Download Site
      ↓
Unknown Installer
      ↓
Potential Problems
```

---

7. Installing Python on Windows

If you are using Windows, download the appropriate Windows installer from Python's official website.

The installer provides the components needed to run Python.

During installation, one option is particularly important:

Add Python to PATH

You should understand what this means before continuing.

---

8. What Is PATH?

PATH is an environment variable used by the operating system to locate executable programs.

Suppose you type:

```
python
```

into a terminal.

Your operating system needs to know:

> "Where is the Python executable?"

PATH helps answer that question.

Conceptually:

```
You type:
python

       ↓

Operating System checks PATH

       ↓

Finds Python executable

       ↓

Starts Python
```

Without appropriate PATH configuration, you may encounter an error such as:

```
'python' is not recognized...
```

or a similar message depending on your system.

---

9. Why "Add Python to PATH" Matters

During Windows installation, you may see an option similar to:

```
☑ Add python.exe to PATH
```

For a beginner development environment, enabling this option makes command-line usage easier.

It allows commands such as:

```
python
```

or:

```
python --version
```

to work from terminals that recognize the updated PATH.

If you miss this option, Python can often still be configured afterward, but it is easier to understand the setting during installation.

---

10. Verify Python Installation

After installation, open a new terminal.

On Windows, you can use:

- Command Prompt
- PowerShell
- Windows Terminal
- the integrated terminal in VS Code

Run:

```
python --version
```

You should see a Python version.

For example:

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

The exact version will depend on the current supported release you installed.

You can also try:

```
python -V
```

`-V` is a shorter form commonly used to request the version.

---

11. Why Should You Open a New Terminal?

Environment variables are loaded when a terminal session starts.

If you install Python or modify PATH while an old terminal is already open, that terminal may not immediately know about the updated configuration.

So if this:

```
python --version
```

does not work immediately after installation:

1. close the terminal
2. open a new terminal
3. run the command again

This simple step solves many beginner installation issues.

---

12. What If `python` Does Not Work?

Do not immediately assume Python is broken.

First check:

```
python --version
```

Then try:

```
py --version
```

Windows installations can provide the Python launcher command `py`.

If one command works and the other does not, Python may still be installed correctly.

For example:

```
python → not recognized

py → Python 3.x.x
```

This tells you that the Python launcher is available even though the `python` command is not configured as expected.

The correct solution depends on your installation and system configuration.

---

13. Python vs `py` on Windows

You may encounter:

```
python
```

and:

```
py
```

They are not simply two different Python languages.

On Windows:

- `python` can invoke the Python executable when it is available on PATH.
- `py` is the Python launcher commonly provided by Python installations.

For example:

```
py --version
```

can show the installed Python version.

You can also run a file through the launcher:

```
py main.py
```

The important lesson is:

The command you use depends partly on your operating system and installation configuration.

---

14. What About `python3`?

On macOS and many Linux environments, you will often see:

```
python3
```

instead of:

```
python
```

For example:

```
python3 --version
```

and:

```
python3 main.py
```

The reason is historical compatibility with systems that had an older `python` command associated with Python 2.

The modern Python ecosystem uses Python 3.

Therefore, you may see three common commands:

```
Windows:
python
py

macOS/Linux:
python3
```

Your exact command depends on your operating system and configuration.

---

15. Your First Python Interpreter Session

Once Python is installed, you can launch the interactive interpreter.

Try:

```
python
```

or on systems where appropriate:

```
python3
```

You may see something similar to:

```
Python 3.x.x ...
>>>
```

The:

```
>>>
```

prompt means Python is waiting for an instruction.

Now type:

```
print("Hello Python")
```

You should see:

```
Hello Python
```

You have just executed Python code.

---

16. What Is the Interactive Interpreter?

The interactive interpreter allows you to execute Python expressions one at a time.

For example:

```
>>> 2 + 3
5
```

Then:

```
>>> 10 * 4
40
```

And:

```
>>> print("Hello")
Hello
```

This environment is useful for:

- experimenting
- testing small ideas
- checking syntax
- learning
- inspecting behavior

However, you should not build an entire application inside the interactive interpreter.

For actual projects, you normally write code in `.py` files.

---

17. Exiting the Python Interpreter

To leave the interactive interpreter, you can use:

```
exit()
```

You can then return to your normal terminal.

The difference is:

```
Terminal
   ↓
Python command
   ↓
Python Interpreter
```

and:

```
Terminal
   ↓
python main.py
```

```
    ↓
Run a Python file
```

Both involve Python, but they are different workflows.

---

18. The Code Editor

You can technically write Python using a basic text editor.

But a dedicated code editor provides useful features such as:

- syntax highlighting
- autocomplete
- error indicators
- file navigation
- search
- debugging tools
- integrated terminal
- extensions
- project management

For this course, **Visual Studio Code** is a practical choice.

Visual Studio Code

---

19. Installing Visual Studio Code

Download Visual Studio Code from its official website.

Install it normally for your operating system.

Once installed, open it.

You should see an interface containing areas for things such as:

- files
- editor
- source control
- extensions
- terminal

Do not worry about understanding every button.

For Python development, we mainly need:

```
Files
+
```

```
Editor
+
Terminal
+
Python Extension
```

---

20. Install the Python Extension

Open the Extensions section in VS Code.

Search for:

```
Python
```

Install the official Python extension published by Microsoft.

This extension adds Python-specific development capabilities to VS Code.

It can provide features such as:

- Python code completion
- debugging
- environment selection
- syntax support
- testing integration
- interpreter management

Remember the distinction:

```
Python
= programming language/runtime

VS Code
= code editor

Python Extension
= VS Code tooling for Python development
```

These are three different things.

---

21. Create Your First Python Project

Do not start by placing Python files randomly across your computer.

Create a dedicated project folder.

For example:

```
python-learning/
```

Inside it:

```
python-learning/
```

Later, your folder might contain:

```
python-learning/
│
├── basics/
├── exercises/
├── projects/
└── notes/
```

For your first program, you can keep it simple.

Create:

```
python-learning/
└── hello.py
```

---

22. Why Project Folders Matter

A project folder gives your code a defined home.

Without organization:

```
Desktop/
├── test.py
├── final.py
├── final2.py
├── test_new.py
├── python_code.py
└── random.py
```

This becomes difficult to manage.

With organization:

```
python-learning/
├── exercises/
├── projects/
└── hello.py
```

You know where things belong.

Professional software development depends heavily on organization.

---

23. Create `hello.py`

Inside your project folder, create:

```
hello.py
```

Add:

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

Save the file.

Now you have a Python source file.

---

24. Run Your Python File

Open the terminal in your project folder.

Run:

```
python hello.py
```

If your environment uses `python3`, run:

```
python3 hello.py
```

On Windows, you can also use:

```
py hello.py
```

You should see:

```
Hello, Python!
```

The process is:

```
hello.py
   ↓
Terminal command
   ↓
Python runtime
   ↓
Execute code
   ↓
Output
```

---

25. What Just Happened?

You created:

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

The file was saved as:

```
hello.py
```

Then you told the Python runtime:

```
Execute hello.py
```

Python read the source code and executed the instruction.

The `print()` function produced output in the terminal.

This is the basic development loop:

```
Write
 ↓
Save
 ↓
Run
 ↓
Observe
 ↓
Change
 ↓
Run Again
```

You will repeat this cycle thousands of times as a developer.

---

26. Editor vs Terminal

Beginners sometimes think the editor runs Python.

The editor is where you write the code.

The Python runtime executes the code.

The terminal provides a way to launch commands.

Think of it as:

```
VS Code
   ↓
You write code

Terminal
```

```
   ↓
You issue command

Python
   ↓
Executes code

Terminal
   ↓
Shows output
```

VS Code can make this workflow feel integrated, but the underlying responsibilities remain different.

---

27. The Integrated Terminal

VS Code includes an integrated terminal.

You can open it from the application and run:

```
python hello.py
```

This is convenient because you do not need to switch between your editor and another terminal window.

You can therefore work like this:

```
VS Code
│
├── Write code
│
├── Browse files
│
└── Open terminal
        ↓
     Run Python
```

---

28. Understanding Your Current Environment

At this point, you have several layers:

```
Operating System
      ↓
Python Installation
      ↓
```

```
Python Interpreter
      ↓
VS Code
      ↓
Python Extension
      ↓
Project Folder
      ↓
.py File
      ↓
Python Program
```

This may seem like a lot.

But each layer solves a different problem.

---

29. What Is the Python Interpreter Selection in VS Code?

When multiple Python installations or environments exist, VS Code needs to know which Python interpreter should be used for your project.

For example, your computer could have:

```
Python A
Python B
Virtual Environment C
```

VS Code needs to know which one should execute your project.

This is why you may see an option called something like:

Python: Select Interpreter

You can choose the Python environment associated with your project.

This becomes especially important when working with virtual environments.

---

30. Why Multiple Python Versions Can Exist

A computer may contain multiple projects.

For example:

```
Project A
requires Python version X

Project B
requires Python version Y
```

Different projects can have different compatibility requirements.

This is one reason professional Python developers use isolated environments.

Do not be afraid if you eventually see multiple Python installations.

The goal is not:

> "My computer must have exactly one Python installation."

The goal is:

> "Each project should use the intended Python environment."

---

31. Virtual Environments: Your First Introduction

A virtual environment is an isolated Python environment associated with a project.

Suppose you create:

```
project-a/
```

You can create an environment for it:

```
project-a/
├── .venv/
└── main.py
```

Another project might have:

```
project-b/
├── .venv/
└── main.py
```

Each environment can have its own installed dependencies.

Conceptually:

```
Project A
   ↓
Environment A
   ↓
Dependencies A

Project B
   ↓
Environment B
   ↓
Dependencies B
```

This prevents many dependency conflicts.

You will study virtual environments properly later in the Python series.

For now, understand **why they exist**.

---

32. Why You Should Not Install Everything Globally

Imagine you install dozens of packages into one global Python environment.

Eventually:

```
Project A
   ↓
Needs package version 1

Project B
   ↓
Needs package version 2
```

Your global environment now has conflicting requirements.

A project-specific virtual environment avoids much of this problem.

The professional pattern is:

```
Project
 ↓
Virtual Environment
 ↓
Project Dependencies
```

---

33. Your First Virtual Environment

Although the detailed topic comes later, it is useful to see the workflow.

From your project folder:

```
python -m venv .venv
```

This creates a virtual environment named:

```
.venv
```

Your project may then look like:

```
python-learning/
│
```

```
├── .venv/
└── hello.py
```

Do not manually modify files inside `.venv`.

The environment is managed by Python tooling.

---

34. Activating a Virtual Environment

The activation command depends on your operating system and shell.

On Windows PowerShell, a commonly used command is:

```
.venv\Scripts\Activate.ps1
```

On Windows Command Prompt:

```
.venv\Scripts\activate
```

On macOS/Linux:

```
source .venv/bin/activate
```

After activation, your terminal often displays the environment name.

For example:

```
(.venv)
```

This indicates that commands are being run inside that environment.

---

35. Why Activation Matters

Suppose you run:

```
pip install requests
```

without understanding which Python environment is active.

You may accidentally install the package somewhere you did not intend.

With a project environment activated:

```
(.venv)
     ↓
pip install requests
     ↓
Install into project environment
```

This makes dependency management much more predictable.

---

36. Do You Need a Virtual Environment for Every Tiny Experiment?

Not necessarily.

For simple learning experiments, you can run Python without creating a new environment every few minutes.

But once you start building a real project or installing third-party dependencies, virtual environments become an important habit.

A useful rule is:

```
Small experiment
→ Python directly

Real project
→ Project-specific environment
```

You will develop a more detailed workflow later.

---

37. Checking Which Python You Are Using

Sometimes you need to know exactly which Python executable your terminal is using.

On Windows, you can use:

```
where python
```

On macOS/Linux:

```
which python
```

If using `python3`:

```
which python3
```

These commands help diagnose situations where multiple Python installations exist.

The result points toward the executable being used.

---

38. Checking Python From Inside Python

You can also inspect Python information from a Python program.

For example:

```
import sys

print(sys.executable)
```

This displays the path of the Python executable currently running the program.

You can also inspect the version:

```
import sys

print(sys.version)
```

This is useful when debugging environment problems.

---

39. A Powerful Debugging Habit

When something doesn't work, do not immediately reinstall everything.

First ask:

```
What exactly is failing?
```

For example:

Problem

```
python hello.py
```

fails.

Instead of randomly changing settings:

```
Check Python version
        ↓
Check Python location
        ↓
Check current folder
        ↓
Check filename
        ↓
Check command
        ↓
Read exact error
```

This is the beginning of systematic debugging.

---

40. Common Problem: Python Command Not Found

You run:

```
python --version
```

and receive an error saying Python is not recognized or cannot be found.

Possible causes include:

- Python is not installed
- Python is installed but not on PATH
- terminal was opened before installation
- another command is being intercepted
- multiple installations exist
- system configuration is incorrect

Do not assume one cause.

Debug systematically.

---

41. Common Problem: Wrong Python Version

You expected:

```
Python 3.x
```

but the terminal reports another version.

First check:

```
python --version
```

Then:

```
where python
```

or:

```
which python
```

You may discover that your terminal is using a different installation than expected.

The solution should target the actual cause instead of blindly reinstalling Python.

---

42. Common Problem: VS Code Uses the Wrong Interpreter

Your terminal says one version, but VS Code behaves as though another version is being used.

This can happen when VS Code has selected a different interpreter.

Check the selected Python interpreter in VS Code.

Make sure it points to the intended environment.

For a project using `.venv`, you generally want VS Code to use:

```
project/.venv/...
```

rather than an unrelated global Python installation.

---

43. Common Problem: `pip` Installs Into the Wrong Environment

You run:

```
pip install package
```

but your Python program cannot import it.

One common cause is:

```
pip
 ↓
Environment A

python
 ↓
Environment B
```

You installed the package into one environment but ran Python from another.

A useful technique is:

```
python -m pip install package
```

This tells the selected Python interpreter to run its associated `pip` module.

Conceptually:

```
Selected Python
      ↓
Its pip
      ↓
Install package
```

This can reduce confusion when multiple environments exist.

---

44. Common Problem: File Not Found

Suppose you create:

```
hello.py
```

but run:

```
python hello.py
```

and receive a file-not-found error.

The issue may simply be that your terminal is currently in a different directory.

Check your current location.

On many shells:

```
pwd
```

On Windows Command Prompt:

```
cd
```

You can also inspect the directory contents.

On Windows:

```
dir
```

On macOS/Linux:

```
ls
```

You want to confirm that:

```
hello.py
```

actually exists in the directory from which you are running the command.

---

45. File Extensions Matter

On Windows, beginners sometimes accidentally create:

```
hello.py.txt
```

instead of:

```
hello.py
```

This can happen when file extensions are hidden in File Explorer.

Make sure the actual filename ends in:

```
.py
```

not:

```
.py.txt
```

A Python source file should have the correct extension.

---

46. Common Problem: Typing Code Into the Wrong Place

There are two very different environments:

Python interpreter

```
>>>
```

Example:

```
>>> print("Hello")
```

Normal terminal

Example:

```
python hello.py
```

If you type:

```
print("Hello")
```

directly into a shell that is expecting operating system commands, it may fail.

Understand which environment you are currently in.

```
Normal Terminal
      ↓
Commands

Python Interpreter
      ↓
Python code
```

---

47. Terminal Navigation Basics

Before building Python projects, you should become comfortable moving between folders.

Useful commands include:

Show current directory

```
pwd
```

or on Windows Command Prompt:

```
cd
```

List files

```
ls
```

or:

```
dir
```

Change directory

```
cd folder-name
```

Move to parent directory

```
cd ..
```

The exact behavior and available options vary between shells, but these basic concepts are universal.

---

48. Why Developers Use the Terminal

The terminal may look intimidating at first.

But it provides direct control over your development environment.

You can use it to:

- run programs
- install packages
- create environments
- run tests
- inspect files
- manage Git
- start servers
- execute scripts
- automate tasks

Later, you will use the terminal constantly.

You do not need to memorize every command.

You need to understand what the command is doing.

---

49. GUI vs Terminal

Suppose you want to run a Python program.

With a GUI workflow:

```
Open VS Code
↓
Open file
↓
Click Run
```

With a terminal workflow:

```
python main.py
```

Both can execute your program.

The terminal becomes especially useful because it gives you precise control and works well with automation, deployment, Git, servers, and CI/CD systems.

That is why developers learn both GUI tools and command-line workflows.

---

50. Your First Proper Python Workspace

A clean beginner workspace could look like:

```
python-learning/
│
├── .venv/
│
├── lessons/
│   ├── hello.py
│   └── variables.py
│
├── exercises/
│   ├── exercise_01.py
│   └── exercise_02.py
│
├── projects/
│   └── first_project/
│
└── README.md
```

You do not need this entire structure on day one.

The important concept is:

Organize code intentionally as the project grows.

---

51. Your First Complete Workflow

Here is the workflow you should now understand:

```
1. Create project folder
        ↓
2. Open project in VS Code
        ↓
3. Create Python file
        ↓
4. Select Python interpreter
        ↓
5. Write code
        ↓
6. Save file
        ↓
7. Open terminal
        ↓
8. Run Python program
        ↓
9. Read output
        ↓
10. Debug if necessary
```

This cycle is the foundation of everyday Python development.

---

52. A Real Example

Suppose you want to build a student marks calculator.

Your project might eventually become:

```
student_marks/
│
├── .venv/
├── main.py
├── calculator.py
└── README.md
```

You start with:

```
marks = 85

print(marks)
```

Then gradually add:

```
Input
↓
Validation
↓
Calculation
↓
Grade Logic
↓
Output
```

The development environment stays the same.

Only your program becomes more sophisticated.

This is important:

The same basic Python environment can support both your first program and much larger applications.

---

53. Installation Is Not Learning

Installing Python is necessary, but it does not make you a Python developer.

You can have:

```
Python installed
VS Code installed
Extensions installed
```

and still not know how to solve programming problems.

The environment is the workshop.

Your programming skills come from:

```
Concepts
+
Practice
+
Problem Solving
```

```
+
Projects
+
Debugging
```

This distinction prevents a common beginner trap: spending too much time configuring tools and too little time writing programs.

---

54. Avoid Configuration Paralysis

Beginners sometimes spend days deciding:

- Which editor?
- Which theme?
- Which extension?
- Which font?
- Which terminal?
- Which operating system?
- Which color scheme?

These decisions are much less important than learning programming.

A practical setup is enough:

```
Python
+
VS Code
+
Python Extension
+
Terminal
+
Git later
```

Start building.

Optimize your tooling when a real need appears.

---

55. Development Environment Checklist

Before moving forward, verify the following.

Python

```
python --version
```

or:

```
python3 --version
```

or on Windows:

```
py --version
```

VS Code

Open the editor successfully.

Python Extension

Install the official Python extension for VS Code.

Project

Create a folder such as:

```
python-learning
```

Python File

Create:

```
hello.py
```

Code

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

Run

```
python hello.py
```

or the appropriate command for your environment.

Result

You should see:

```
Hello, Python!
```

If all of these work, your environment is ready.

---

56. Five Minute Setup Challenge

Complete these steps without copying the exact workflow blindly.

Step 1

Create a folder:

```
python-practice
```

Step 2

Open it in VS Code.

Step 3

Create:

```
main.py
```

Step 4

Write:

```
print("I am learning Python")
```

Step 5

Run the program from the terminal.

Step 6

Change the message to:

```
print("I can run my first Python program")
```

Step 7

Run it again.

The goal is to become comfortable with the cycle:

```
Edit → Save → Run → Observe
```

---

57. Practice Exercises

Exercise 1 — Environment Verification

Run the appropriate command to determine your Python version.

Write down:

```
Python Version:
Python Executable:
Operating System:
Code Editor:
```

Exercise 2 — Interpreter Experiment

Open the Python interpreter and test:

```
2 + 8
```

Then:

```
10 * 5
```

Then:

```
print("Python works")
```

Observe how Python responds immediately.

Exercise 3 — Create Three Files

Create:

```
hello.py
about.py
math_test.py
```

Put a different simple Python statement in each file.

Run each file from the terminal.

Exercise 4 — Deliberate Error

Create:

```
print("Hello"
```

Run it.

Read the error message.

Do not immediately search for the answer.

First identify:

- What line failed?
- What kind of problem does Python report?
- What part of your code appears incomplete?

Then fix it.

This is your first debugging exercise.

Exercise 5 — Directory Thinking

Create:

```
python-practice/
├── lessons/
└── exercises/
```

Put your lesson files in `lessons`.

Put your exercise files in `exercises`.

Then use the terminal to move between the directories.

---

58. Mini Quiz

Question 1

What is the purpose of Python?

A. To edit images

B. To execute Python programs and provide the programming environment

C. To replace Windows

D. To create databases automatically

Answer: B

Question 2

What does PATH help the operating system do?

A. Store Python variables

B. Find executable programs

C. Write Python code

D. Create databases

Answer: B

Question 3

What is VS Code?

A. A Python implementation

B. A database

C. A code editor

D. A Python package

Answer: C

Question 4

What does this command generally do?

```
python main.py
```

A. Deletes `main.py`

B. Opens a database

C. Executes `main.py` using Python

D. Installs Python

Answer: C

Question 5

Why are virtual environments useful?

A. They make your monitor faster

B. They isolate project-specific Python dependencies

C. They replace Python

D. They remove the need for code

Answer: B

---

59. Interview Questions

What is PATH?

PATH is an operating system environment variable that contains locations where executable programs can be found.

Why might `python` not work after installing Python?

Possible reasons include PATH configuration, an old terminal session, multiple Python installations, or operating-system-specific command configuration.

What is the difference between VS Code and Python?

VS Code is a code editor. Python is a programming language and runtime ecosystem.

What is the Python interpreter?

It is the software that executes Python code.

Why do developers use virtual environments?

They isolate a project's Python interpreter and dependencies from other projects, reducing dependency conflicts.

Why can `python` and `python3` both exist?

Operating systems and installations may expose different commands for invoking Python. `python3` is commonly used on macOS/Linux, while Windows may provide `python` and `py`.

Why should you use the terminal?

The terminal provides direct control over running programs, managing environments and packages, testing, Git, deployment, and other development tasks.

---

60. Environment Cheat Sheet

Check Python

```
python --version
```

or:

```
python3 --version
```

or:

```
py --version
```

Start Python

```
python
```

Run a file

```
python main.py
```

Create virtual environment

```
python -m venv .venv
```

Activate on Windows PowerShell

```
.venv\Scripts\Activate.ps1
```

Activate on Windows Command Prompt

```
.venv\Scripts\activate
```

Activate on macOS/Linux

```
source .venv/bin/activate
```

Find Python on Windows

```
where python
```

Find Python on macOS/Linux

```
which python
```

Check executable from Python

```
import sys

print(sys.executable)
```

Check Python version from Python

```
import sys

print(sys.version)
```

Exit interpreter

```
exit()
```

---

61. What You Should Understand Before Moving On

You should now understand the difference between:

```
Python
Programming language/runtime

VS Code
Code editor

Terminal
Command-line interface

Python Extension
Python tooling inside VS Code

.py
Python source file

Interpreter
Software that executes Python

PATH
Operating-system mechanism for locating executables
```

```
Virtual Environment
Isolated project environment
```

These terms will appear repeatedly throughout your Python journey.

---

62. The Development Loop You Will Use Again and Again

Almost every Python project begins with some variation of:

```
Think
 ↓
Write
 ↓
Run
 ↓
Observe
 ↓
Debug
 ↓
Change
 ↓
Run Again
```

As your projects become more advanced, additional stages appear:

```
Think
 ↓
Design
 ↓
Implement
 ↓
Test
 ↓
Debug
 ↓
Refactor
 ↓
Deploy
 ↓
Monitor
```

Your development environment supports this entire process.

---

63. The Bigger Picture

You started with:

```
"What is Python?"
```

Now you have:

```
Computer
   ↓
Python installed
   ↓
Python interpreter
   ↓
VS Code
   ↓
Project folder
   ↓
Python file
   ↓
Terminal
   ↓
Program execution
```

This is your first real development environment.

The next step is not another installation.

It is learning how Python code is actually executed and understanding the relationship between:

```
Python Source Code
        ↓
Execution
        ↓
Output
```

That understanding will make everything that follows easier.

Key Takeaways

- Install Python from an official source.
- Use a current supported Python release unless a project requires another version.
- Understand what PATH does instead of treating it as a mysterious setting.
- Verify your Python installation from the terminal.

- Learn the difference between `python`, `python3`, and `py`.
- Use a dedicated code editor such as VS Code.
- Install Python tooling for your editor.
- Keep Python projects inside organized project folders.
- Learn to run `.py` files from the terminal.
- Understand the difference between the terminal and Python interpreter.
- Use virtual environments for real projects with dependencies.
- When something fails, read the exact error before changing your setup.
- Your development environment is the toolset; your programming ability comes from learning, practicing, building, and debugging.

Visit haas.dev for more resources and guides.

haas.dev

https://dev-roast-app.vercel.app