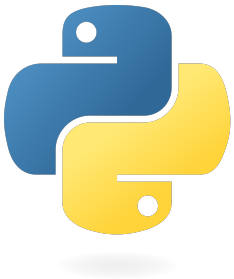
The Bare Minimum of Python

A 4-Week Hands-on Learning Guide

Kshitij Dahal
geoskhitij@gmail.com



May 22, 2025

Contents

Week 1: Understanding Programming and Python

> *"Learning Python is not about learning Python itself, but learning how it works."*

1 What is Programming?

Programming is giving instructions to a computer. Just like we follow a recipe to cook food, computers follow programs to do tasks.

A program is a list of steps written in a language the computer understands.

Programming helps us solve problems using computers. It helps in doing tasks faster, better, and in a repeatable way.

2 Why Learn Programming?

We use programming to:

- Build websites
- Analyze data
- Automate tasks
- Create apps and games

Learning programming helps you become a problem-solver in the tech world.

3 What is Python?

Python is a popular programming language. It is used in many fields:

- Data science
- Machine learning
- Web development
- Automation

Python is easy to read and simple to write. That is why many beginners start with Python.

4 Think of Python as a Home

Imagine Python as a home. Inside the home, there are many tools:

- A rice cooker for cooking rice
- A washing machine for cleaning clothes
- A fridge for keeping things cool

Each tool does a job. In Python, these tools are called **libraries**. A library is a group of code that does something useful.

You don't have to build every tool from scratch. Python/community gives you these tools to use.

5 What is an Ecosystem?

The ecosystem is all the tools, packages, and environments that work together in Python.

If your company is doing data analysis, they might use tools like:

- `pandas` for working with data
- `numpy` for numbers
- `matplotlib` for charts

You need to learn what tools are used in your work. Not just the Python language.

6 Why the Ecosystem Matters

You should not only learn Python syntax. You must learn how the whole system works.

That includes:

- Setting up your environment (we will do that in Week 2)
- Installing packages
- Fixing errors and missing tools
- Understanding how your team uses Python

7 Conclusion

In Week 1, we learned:

- What programming is
- What Python is
- How Python works like a home with tools
- Why learning the ecosystem is important

Next week, we will learn how to create our own Python environment using tools like Conda and Docker.

8 A Short History of Programming

Programming has been around for a long time. It started even before modern computers.

Early Days

In the 1800s, a woman named Ada Lovelace wrote the first idea of a computer program. She worked with Charles Babbage, who was building a machine.

In the 1940s, computers were big and slow. Programs were written using switches and punched cards. It was hard and took a lot of time.

First Programming Languages

In the 1950s and 1960s, people made languages to make programming easier. Some early languages were:

- **Fortran** — used for science and math
- **COBOL** — used in business and banking

These were still hard to learn, but better than using switches!

Modern Programming Languages

Later, more friendly languages were created. These include:

- **C** — powerful and fast
- **Java** — runs on many systems
- **Python** — simple and clean

Python became popular because it is easy to read. It helps you focus on solving problems instead of writing long code.

Now and the Future

Today, we write programs for many things:

- Websites
- Phones
- Data science
- Artificial Intelligence (AI)

Programming is part of daily life. It helps us build smart tools and apps.

Python is now used in schools, research, and big tech companies.

This is why learning Python is a great first step.

"The best time to start learning was 10 years ago. The second-best time is today."

Week 2: Setting Up Your Python Environment

"You will spend more time fixing your environment than writing code."

9 Why Do We Need Environments?

Every project may need different tools (libraries). If you install everything in one place, things may break.

An **environment** keeps your project clean. It holds:

- Python version
- Libraries you need
- Configurations

Think of it like a lunchbox. Everything needed for one project stays inside.

10 Conda: Easy Environment Manager

Conda helps you:

- Create Python environments
- Install packages
- Manage libraries safely

You can install Conda using:

- **Anaconda** (big, includes many tools)
- **Miniconda** (small, just the basics)

Common Conda Commands

```
# Create a new environment
conda create -n myenv python=3.10

# Activate the environment
conda activate myenv

# Install a library
conda install numpy

# See all environments
conda env list

# Deactivate environment
conda deactivate
```

11 Installing Conda

Go to: `https://docs.conda.io/en/latest/miniconda.html`

Download Miniconda for your system. Follow the instructions to install.

After that, open your terminal (or Anaconda Prompt on Windows).

Try:

```
conda --version
```

If you see a version number, it works!

12 Using Docker (Optional but Powerful)

Docker lets you create containers. These are like mini-computers inside your computer.

Why use Docker?

- Same code runs everywhere
- No version problems
- Easy to share with teams

Example: Your code works on your machine, but not on your friend's. With Docker, it will work the same on both.

Basic Docker Terms

- **Image** — a snapshot of code and environment
- **Container** — a running version of an image

Basic Docker Commands

```
# Check if Docker is installed
docker --version

# Run a Python container
```

```
docker run -it python:3.10

# Build a custom container (using Dockerfile)
docker build -t myapp .

# List all containers
docker ps -a
```

13 Choosing Between Conda and Docker

If you're just starting: **Use Conda**. It is easier.

Use Docker when:

- You want full control
- You work with teams
- You need the same setup across systems

14 pip vs conda: Package Managers

There are two common tools for installing Python libraries:

- **pip**
- **conda**

They both install packages, but they work differently.

What is pip?

`pip` is the Python Package Installer.

It downloads packages from a place called PyPI (Python Package Index).

Example:

```
pip install pandas
```

`pip` works well for pure Python packages. But sometimes, it has problems with packages that need system libraries (like C or C++ code).

These problems are called **dependency issues**. It means:

- One library needs another library
- Or different versions of libraries conflict

You may face errors like:

```
ERROR: Failed building wheel for ...
```

What is conda?

`conda` is both a package manager and an environment manager.

It installs packages from a Conda channel — a place where packages are stored.

Conda solves dependency issues better because it installs:

- Python code
- System libraries
- Tools

Conda Channels

Channels are sources of Conda packages.

- **defaults** — the main channel from Anaconda
- **conda-forge** — a community-driven channel with many packages

If something is not available in defaults, try conda-forge.

Example:

```
conda install -c conda-forge jupyterlab
```

The `-c` flag means "use this channel."

Using pip inside Conda (carefully)

You can use `pip` inside a Conda environment, but be careful.

If you mix pip and conda too much, it may break your environment.

```
conda install numpy
pip install some-small-library
```

Use `conda` when possible. Use `pip` only if the package is not available in conda.

15 Conclusion

Now you know:

- Why environments are important
- How to use Conda

- How to try Docker

Next week, we will learn Git and GitHub. These help you save, track, and share your code.

16 Crash Course: Using the Shell (Command Line)

The **shell** (or terminal) is a way to talk to your computer using text.

It looks like a black screen, but it is very powerful.

You can use it to:

- Create folders and files
- Run programs
- Install packages
- Navigate your system

Opening the Shell

- **Windows:** Use `Anaconda Prompt` or `Command Prompt`
- **macOS:** Use `Terminal`
- **Linux:** Use your system's terminal app

Basic Shell Commands

- `pwd` — show current location
- `ls` — list files and folders

- `cd foldername` — go into a folder
- `cd ..` — go back one level
- `mkdir myfolder` — make a new folder
- `touch file.py` — create a new file (macOS/Linux)
- `echo Hello > file.txt` — write to a file
- `rm file.txt` — delete a file

Running Python from Shell

Once your environment is ready, you can run Python scripts like this:

```
python myscript.py
```

Or open the Python interpreter:

```
python
```

Running Jupyter Notebook from Shell

If you have Jupyter installed:

```
jupyter notebook
```

This will open a browser window to start coding in notebooks.

Why Learn the Shell?

Many tools (like Conda, Git, and Docker) work best in the shell. Even Python itself is often run from there.

Knowing just a few commands helps you work faster and smarter.

Week 3: Version Control

> *"You will not find and understand the code written by yourself after one month — unless you use version control."*

17 Why Version Control?

When you're writing code, things change a lot. Sometimes, you break things. Sometimes, you want to go back.

Version control helps you:

- Track changes in your code
- Save versions of your project
- Work with a team without messing up each other's code

The most popular version control system today is **Git**.

18 What is Git?

Git is a tool that helps you keep snapshots of your code over time.

With Git, you can:

- Go back to an earlier version
- See who changed what
- Create branches to try new ideas

Installing Git

Go to: `https://git-scm.com/downloads`

Download and install Git for your system. After installation, try:

```
git --version
```

19 Basic Git Workflow

Here's a simple Git workflow:

1. Create a folder or go to your project folder
2. Initialize Git:

   ```
   git init
   ```

3. Make changes to your code
4. Check status:

   ```
   git status
   ```

5. Add files to staging:

   ```
   git add filename.py
   ```

   Or add everything:

   ```
   git add .
   ```

6. Commit the changes:

   ```
   git commit -m "Add initial version"
   ```

Now your changes are saved in Git!

20 What is GitHub?

GitHub is a website where you can store your Git projects online.

With GitHub, you can:

- Back up your code
- Share code with others
- Work on open-source projects

Go to: `https://github.com` and create an account.

Connecting Git to GitHub

1. Create a new repository on GitHub.
2. Copy the URL (it looks like `https://github.com/yourname/repo.git`).
3. Connect your local Git to GitHub:

   ```
   git remote add origin https://github.com/yourname/repo.git
   ```

4. Push your code:

   ```
   git push -u origin master
   ```

21 Common Git Commands

```
# Check current branch and status
git status
```

```
# Add files to staging
git add filename.py

# Commit your changes
git commit -m "your message"

# See commit history
git log

# Create a new branch
git checkout -b new-feature

# Switch branches
git checkout main

# Push to GitHub
git push

# Pull updates from GitHub
git pull
```

22 Best Practices

- Commit small changes often
- Write clear commit messages
- Don't commit secrets like passwords
- Use branches for new features

23 Conclusion

Now you know how to:

- Use Git to track your code
- Save your project on GitHub
- Work with others using version control

Next week, we will finally start writing Python code and use all the tools we've learned so far!

Week 4: Writing Python Code

> *"You don't need to write the code — you just need to understand what exists, so you can prompt ChatGPT better."*

24 Let's Write Python Code!

This week, we will start writing actual Python code.

You do not need to memorize everything. You will use tools like **ChatGPT** to help you write code.

Your job is to:

- Understand what the code does
- Run it in your environment
- Try changing things and see what happens

25 Basic Python Concepts

Before we build something, let's learn the basic pieces of Python.

Numbers

Python can do math easily:

```
print(5 + 3)
print(10 / 2)
```

```
print(4 * 6)
print(7 - 2)
```

Strings (Text)

Strings are words or sentences in quotes:

```
print("Hello, world!")
name = "Alice"
print("My name is " + name)
```

Variables

Variables store values like numbers or strings:

```
a = 10
b = 5
result = a + b
print(result)
```

Input and Output

You can ask users for input:

```
name = input("What is your name? ")
print("Hello " + name)
```

If Conditions

This lets your code make decisions:

```
num = 5
if num > 3:
```

```
    print("Greater than 3")
else:
    print("3 or less")
```

Functions

Functions are blocks of code you can reuse:

```
def greet(name):
    print("Hello, " + name)

greet("Kshitij")
```

26 Let's Build a Simple Calculator

Now let's build a calculator that can add, subtract, multiply, and divide.

We'll use everything we just learned.

Simple Calculator Code

```
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

def multiply(x, y):
    return x * y
```

```
def divide(x, y):
    if y == 0:
        return "Error: Cannot divide by zero"
    return x / y

print("Select operation:")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")

choice = input("Enter choice (1/2/3/4): ")
a = float(input("Enter first number: "))
b = float(input("Enter second number: "))

if choice == '1':
    print("Result:", add(a, b))
elif choice == '2':
    print("Result:", subtract(a, b))
elif choice == '3':
    print("Result:", multiply(a, b))
elif choice == '4':
    print("Result:", divide(a, b))
else:
    print("Invalid input")
```

27 Try Modifying It!

Change things and try:

- Adding new operations (like power or remainder)
- Making a loop so the user can try again
- Making the output nicer

28 Use ChatGPT to Help You

If you're stuck, ask ChatGPT:

- `"Write a Python function to calculate the square root"`
- `"Make my calculator run in a loop"`
- `"Explain this Python error message"`

Use it as a helper. You focus on understanding.

29 Conclusion

Now you've written your first Python program!

You learned:

- How to use variables, inputs, functions
- How to build a calculator
- How to use ChatGPT as a coding assistant

From here, you can start building small tools and projects. Keep experimenting and stay curious!

Tips That Took Others 10 Years to Learn

These are things many developers wish they had learned earlier. If you remember these, you'll grow much faster.

- **Don't memorize code, understand how things work.**
- **Always search your errors.** Copy the error message and Google it. Someone else has faced it too.
- **Read the documentation.** It's the most reliable source of truth.
- **Use ChatGPT or tools like it, not as a crutch, but as a pair of eyes.**
- **Take breaks.** Debugging for 2 hours can sometimes be fixed by a 10-minute walk.
- **Keep your environment clean.** Use Conda, virtualenv, or Docker to avoid "it works on my machine" problems.
- **Build a portfolio.** Share your projects on GitHub. Add them to a simple personal site. Show the world you know stuff.
- **Explain things to others.** Teaching is the best way to learn deeply.
- **Don't rush coding.** Slow is smooth. Smooth becomes fast.
- **Use version control early.** Even solo projects benefit from Git. It's your project's time machine.
- **Done is better than perfect.** Start small. Ship things. Improve later.