

PRACTICAL COURSE • CLASSROOM READY • 30 CHAPTERS



30-DAY PYTHON

LEARNING PLAN

Learn a little every day • build a skill for life

2026 STUDENT EDITION

FREE PREVIEW • DAYS 1-3

```
day_1.py

print("Hello, World!")
>>> Hello, World! |
```

Fundamentals

Control Flow

Data & Functions

Files & OOP

5 Projects

✓ Zero to Confident Coder ✓ 150+ Practice Exercises ✓ Real Mini Projects

AUTHOR

Gaurav Malviya



CONSISTENCY > MOTIVATION

Index

Welcome to the Course	3
------------------------------------	----------

WEEK 1 · Python Fundamentals

Day 1 Introduction to Python	6
Day 2 Setup & First Program	8
Day 3 Variables & Data Types	10
Day 4 Basic Operators	FULL EDITION
Day 5 Input & Output	FULL EDITION
Day 6 Type Casting	FULL EDITION
Day 7 Mini Project & Review	FULL EDITION

WEEK 2 · Control Flow

Day 8 If Statements	FULL EDITION
Day 9 If-Else & Elif Statements	FULL EDITION
Day 10 Loops (For, While)	FULL EDITION
Day 11 Break, Continue & Pass	FULL EDITION
Day 12 Nested Loops	FULL EDITION
Day 13 Practice Problems	FULL EDITION
Day 14 Mini Project & Review	FULL EDITION

WEEK 3 · Data Structures & Functions

Day 15 Lists	FULL EDITION
Day 16 Tuples	FULL EDITION
Day 17 Sets	FULL EDITION
Day 18 Dictionaries	FULL EDITION
Day 19 Functions Basics	FULL EDITION
Day 20 Return Statement	FULL EDITION
Day 21 Mini Project & Review	FULL EDITION

WEEK 4 · Modules, Files & OOP

Day 22 Modules & Import	FULL EDITION
Day 23 File Handling (Basics)	FULL EDITION
Day 24 Errors & Exceptions	FULL EDITION
Day 25 OOP Introduction	FULL EDITION
Day 26 Classes & Objects	FULL EDITION
Day 27 Practice Problems	FULL EDITION
Day 28 Mini Project & Review	FULL EDITION

FINAL PUSH · Projects

Day 29 Build a Small Project	FULL EDITION
Day 30 Final Project & Review	FULL EDITION

APPENDICES

App. A Quick Reference Card	FULL EDITION
App. B Common Errors & Fixes	FULL EDITION
App. C 30-Day Plan at a Glance	FULL EDITION

Welcome to the Course

This book takes an absolute beginner to a confident Python programmer in one month. Each day is one chapter, and every chapter follows the same rhythm: **Understanding** (the idea in plain language), **Definition** (exam-ready wording), **Syntax**, **Examples with output**, **Practice Exercises**, and **Key Points** for revision.

How the Course Is Divided



Your 30-Day Learning Journey



For Teachers

Teach one chapter per day. Every 7th day is a **Mini Project & Review** day with a complete solution, extension challenges, and a marking rubric. The recommended session plan:

Recommended 60-Minute Class Plan (every chapter)

Concept & Definitions 20 min	Live Coding Examples 20 min	Student Practice 15 min	Recap & Key Points 5 min
---------------------------------	--------------------------------	----------------------------	-----------------------------

For Students

Do not skip days — each chapter builds on the last. Type every example yourself and run it before reading the explanation. Reading code is not learning code; typing it, breaking it, and fixing it is. All you need is a computer with **Python 3.10+** (free from python.org), an editor such as IDLE, Thonny, or VS Code, and a notebook.

Days 1-7

WEEK 1: Python Fundamentals

Install Python, write your first programs, and master variables, data types, operators, input/output, and type casting. By Day 7 students will build their first mini project.

Understanding

Imagine you want to give instructions to a very obedient but very literal servant. He will do *exactly* what you say — nothing more, nothing less. A computer is that servant, and a **programming language** is the language you use to give it instructions.

Python is one of the easiest programming languages for humans to read and write. Its instructions look almost like English sentences. That is why Python is the world's most popular first language — and also why it powers Google, YouTube, Instagram, NASA projects, and modern Artificial Intelligence.

Why learn Python?

- **Simple syntax** — less code to do more work
- **Free and open source** — anyone can download and use it
- **Huge job demand** — data science, web development, automation, AI/ML
- **Massive community** — millions of learners and free resources

Definition

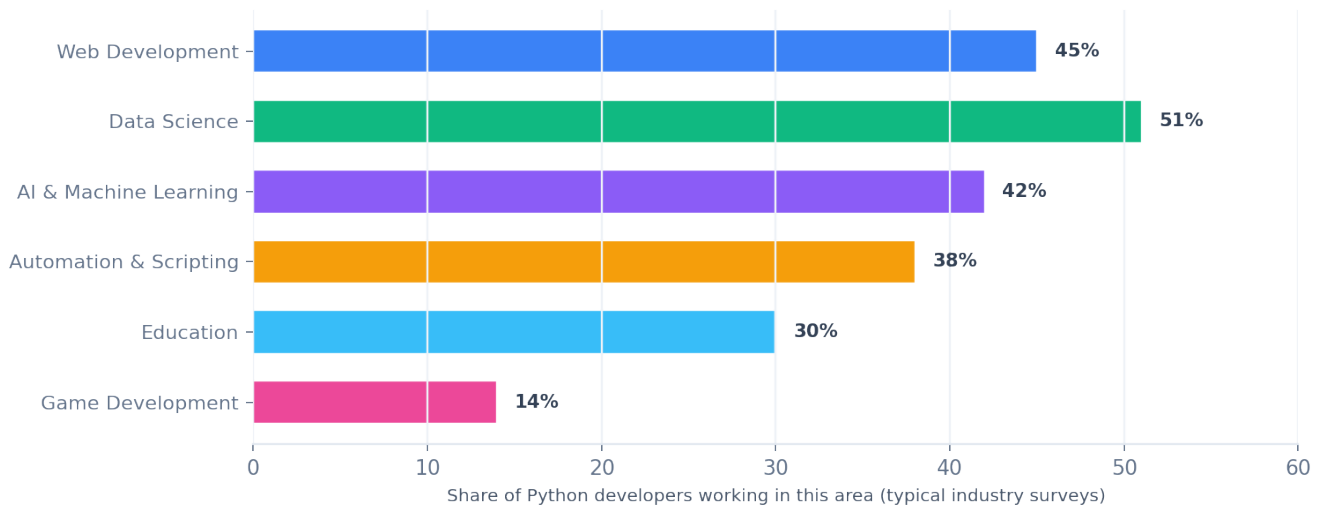
Python is a high-level, interpreted, general-purpose programming language created by **Guido van Rossum** and first released in **1991**. “High-level” means it is close to human language; “interpreted” means the code runs line by line without needing to be compiled first.

Key Terms

- **Program** — a set of instructions given to a computer to perform a task.
- **Interpreter** — the software that reads Python code line by line and executes it.
- **IDE (Integrated Development Environment)** — software where you write, run, and debug code (e.g., IDLE, VS Code, Thonny).

Where Python Is Used

Where Python Is Used in the Real World



Field	Real Example
Web development	Instagram, YouTube backends
Data science	Analyzing sales, weather, cricket statistics
AI & Machine Learning	ChatGPT-style models, face recognition
Automation	Auto-sending emails, renaming 1,000 files at once
Game development	Simple 2D games with Pygame
App scripting	Blender, GIS software

Classroom Activity

1. Discuss: name 5 apps you used today. Which ones might use Python?
2. Visit **python.org** and explore the official website.
3. Write in your notebook: three reasons *you* want to learn Python.

Key Points

- Python is high-level, interpreted, and beginner-friendly.
- Created by Guido van Rossum in 1991.
- Used in web, data science, AI, automation, and more.
- Python code is executed line by line by the interpreter.

Understanding

Before a carpenter builds furniture, he sets up his workshop. Today we set up our programming workshop: we install Python, meet the editor, and write our first program.

The tradition in programming is that your first program prints the words *Hello, World!* on the screen. In many languages this takes 5-10 lines. In Python, it takes **one**.

Installing Python (Windows)

1. Go to **python.org** → **Downloads** and download the latest Python 3 version.
2. Run the installer. **Important:** tick the checkbox **"Add Python to PATH"** before clicking Install.
3. Verify: open Command Prompt and type `python --version`. You should see something like `Python 3.12.4`.

On most Linux systems and macOS, Python 3 is pre-installed — check with `python3 --version`.

Two Ways to Run Python

1. **Interactive mode (the shell/REPL)** — type `python` in the terminal. You get a `>>>` prompt. Each line runs immediately. Great for quick experiments.
2. **Script mode** — write code in a file ending with `.py` (e.g., `hello.py`) and run it with `python hello.py`. This is how real programs are written.

Your First Program

```
# hello.py - my first Python program
print("Hello, World!")
print("Welcome to the 30-Day Python Course")
```

Output:

```
Hello, World!
Welcome to the 30-Day Python Course
```

Understanding the Code

- `print()` is a **built-in function** — a ready-made command that displays text on the screen.
- The text inside quotes is called a **string**.

- The line starting with `#` is a **comment** — Python ignores it. Comments are notes for humans.

```
# This is a single-line comment  
print("Comments help explain code") # comment can follow code too
```

Common Beginner Errors

```
print("Hello)      # Error: missing closing quote  
Print("Hello")    # Error: Python is case-sensitive; 'Print' is not defined  
print "Hello"      # Error: parentheses are required in Python 3
```

Read error messages carefully — they tell you the line number and the problem. Errors are teachers, not enemies.

Practice Exercises

1. Write a program that prints your name, your city, and your goal in three separate lines.
2. Write a program that prints a triangle using `*` characters.
3. Deliberately make each of the three errors above, run them, and write down what error message Python shows.
4. Print the following exactly: `She said, "Python is easy!"` (Hint: use single quotes outside.)

Key Points

- Install Python from python.org; always tick “Add to PATH” on Windows.
- `print()` displays output on screen.
- `#` starts a comment; Python ignores it.
- Python is case-sensitive: `print` works, `Print` does not.

Understanding

Think of a variable as a **labeled container**. In your kitchen you have containers labeled “Sugar”, “Rice”, “Tea”. You can look inside them, refill them, or replace the contents — the label stays.

In Python, a variable is a name that refers to a value stored in memory:

```
age = 15
```

Here `age` is the label, `15` is the value, and `=` means “store this value under this name” (it is called the **assignment operator**, not “equals”).

Definition

A **variable** is a named reference to a value stored in the computer’s memory. A **data type** describes what kind of value a variable holds (number, text, true/false, etc.), which decides what operations can be performed on it.

Rules for Naming Variables

- Must start with a letter or underscore: `name`, `_count` ✓
- Cannot start with a digit: `2name` ✗
- Can contain letters, digits, underscore: `student_1` ✓
- No spaces or symbols: `my name`, `roll-no` ✗
- Case-sensitive: `Age` and `age` are different variables
- Cannot be a Python keyword: `if`, `for`, `class`, `print` ✗

Good practice: use meaningful names — `total_marks` is better than `tm`.

Basic Data Types

Type	Name	Example	Use
<code>int</code>	Integer	<code>25</code> , <code>-3</code> , <code>0</code>	Whole numbers
<code>float</code>	Floating point	<code>3.14</code> , <code>-0.5</code>	Decimal numbers
<code>str</code>	String	<code>"Ravi"</code> , <code>'A'</code>	Text
<code>bool</code>	Boolean	<code>True</code> , <code>False</code>	Yes/No logic
<code>NoneType</code>	None	<code>None</code>	"No value yet"

Examples

```
name = "Aarav"      # str
age = 14             # int
height = 5.2         # float
is_student = True    # bool

print(name, age, height, is_student)
print(type(name))
print(type(age))
print(type(height))
print(type(is_student))
```

Output:

```
Aarav 14 5.2 True
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

The `type()` function tells you the data type of any value — very useful for learning and debugging.

Variables can change value (and even type):

```
score = 10
print(score)      # 10
score = 25
print(score)      # 25
score = "high"    # allowed in Python (dynamic typing)
print(score)      # high
```

Multiple assignment:

```
x, y, z = 1, 2, 3
a = b = c = 0
print(x, y, z, a, b, c)  # 1 2 3 0 0 0
```

Practice Exercises

1. Create variables for your name, class, roll number, and percentage; print them with `type()` of each.
2. Which of these are valid names? `1st_rank`, `first_rank`, `_temp`, `my var`, `for`
3. Swap the values of two variables `a = 5` and `b = 10` (Hint: `a, b = b, a`).
4. Create a variable, print it, change its value twice, and print after each change.

Key Points

- A variable is a named container for a value; `=` assigns a value.
- Core types: `int`, `float`, `str`, `bool`, `None`.
- `type()` reveals the data type.
- Python variables can change type — this is called dynamic typing.

END OF FREE PREVIEW

Enjoyed the first 3 days? There are 27 more waiting.

The full **30-Day Python Learning Plan** continues with control flow, loops, data structures, functions, files, OOP — and five complete projects: the Report Card Generator, ATM Simulator, Contact Book, Quiz Master, and the Library Management System capstone.

Inside the full edition: 30 daily chapters · 150+ practice exercises · charts & flowcharts · teacher rubrics for classroom use · complete project solutions.

Get the full edition: contact **Gaurav Malviya**.

From Hello, World! to Real Projects — in 30 Days

A complete, classroom-ready Python course. One chapter a day: a concept explained in plain language, an exam-ready definition, working examples with output, and practice exercises — finishing every week with a real mini project.

- ✓ 30 structured daily chapters
- ✓ 150+ practice exercises & lab days
- ✓ 5 complete projects: Report Card, ATM, Contact Book, Quiz App, Library System
- ✓ Charts, flowcharts & revision cards in every week
- ✓ Teacher rubrics for classroom assessment

Gaurav Malviya

Educator & Software Consultant



CONSISTENCY > MOTIVATION