Python Zero to Hero — 2026 Complete Course

Python Zero to Hero — The Complete 2026 Beginner Course

TechSparking Free Course — 12 chapters · 30 lessons · 3 capstone projects

---

How to use this course

This is a full beginner curriculum, not a quick reference. Work through the chapters in order, type every code example yourself, and complete the exercises before moving on. The capstone projects at the end are the most important part — they turn knowledge into skill. Expect 20–40 hours total over 4–8 weeks. Pair this PDF with our free article *Learn Python in 30 Days: The Complete Free Daily Roadmap* for a day-by-day plan.

Chapter 1 — Setting Up Python

Lesson 1.1 — What Python is and why it matters

Python is a general-purpose programming language created by Guido van Rossum in 1991. It was designed to be readable: code in Python looks closer to plain English than almost any other language. Today it powers:

- Web applications (Django, FastAPI, Flask)
- Data science and machine learning (pandas, scikit-learn, PyTorch)
- Automation and scripting (the "glue" of the IT world)
- Cybersecurity tooling (Scapy, Nmap wrappers, exploit frameworks)
- Artificial intelligence (most AI models are trained with Python)

Lesson 1.2 — Installing Python on Windows, macOS and Linux

Windows: Download the installer from python.org, check the box **"Add Python to PATH"** during install, then verify by opening Command Prompt and typing:

```
python --version
```

macOS: Install via the Homebrew package manager:

```
brew install python
```

Linux (Debian/Ubuntu):

```
sudo apt update && sudo apt install python3
```

Alternative — no install needed: For the first three chapters you can use an online shell like Replit or Google Colab. Many beginners prefer to start coding immediately, then install Python locally in week two.

Lesson 1.3 — Your first program

Create a file called `hello.py` with this content:

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

Run it:

```
python hello.py
```

You should see `Hello, TechSparking!` printed. That's it — you're a programmer now.

Exercises — Chapter 1

1. Print your name and your favorite tech gadget in two separate lines.
2. Print a short poem (four lines) using four `print()` calls.
3. Create a second file that prints "I will finish this course" 3 times.

Chapter 2 — Variables and Data Types

Lesson 2.1 — Variables

A variable is a named box that holds a value. Python lets you create one with =:

```
site_name = "TechSparking"
```

```
articles = 30
rating = 4.8
is_free = True
```

Lesson 2.2 — The core data types

Type	Example	What it holds
`int`	`42`	Whole numbers
`float`	`3.14`	Decimal numbers
`str`	`"hello"`	Text
`bool`	`True`	True or False
`list`	`[1, 2, 3]`	Ordered collection
`dict`	`{"key": "value"}`	Key-value pairs

Check any variable's type:

```
print(type(articles))  # <class 'int'>
```

Lesson 2.3 — String basics

```
name = "Tech Sparking"
print(name.upper())        # TECH SPARKING
print(name.lower())        # tech sparking
print(len(name))           # 13
print(name[0:4])           # Tech
```

Exercises — Chapter 2

1. Create variables for your name, age, and favorite programming language; print a sentence that uses all three.
2. Compute and print the average of three numbers stored in variables.
3. Create a list of your five favorite apps and print the first and last items.

Chapter 3 — Control Flow

Lesson 3.1 — Conditionals with if/elif/else

```
score = 78
if score >= 90:
    print("A")
elif score >= 75:
    print("B")
else:
    print("Pass")
```

Lesson 3.2 — Comparison and logical operators

- == equal · != not equal · > < >= <=
- `and` · `or` · `not`

```
age = 17
parent_ok = True

if age >= 18 or parent_ok:
    print("You can take this online course alone")
```

Lesson 3.3 — Loops: while and for

```
# while loop — repeat until a condition changes
count = 0
while count < 5:
    print("Looping:", count)
    count += 1

# for loop — repeat for each item
apps = ["Chrome", "VSCode", "Spotify"]
for app in apps:
    print("App:", app)
```

Range helper

```
for i in range(1, 6):
    print(i)  # prints 1 2 3 4 5
```

Exercises — Chapter 3

1. Write a program that tells someone if they can get a driving license (age 18) and where they live (allowed in their region).
2. Print all even numbers from 1 to 100 using a loop.
3. Build a simple number guessing game: the program picks a secret number, the user guesses until correct, and gets warmer/colder hints.

Chapter 4 — Lists and Dictionaries

Lesson 4.1 — Lists in depth

```
tools = ["Python", "Git", "Docker"]
tools.append("Linux")          # add to end
tools.insert(0, "VSCode")      # add at position
tools.remove("Docker")         # remove by value
last = tools.pop()             # remove and return the end
print(len(tools))
print(tools[1])
```

List slicing

```
numbers = [10, 20, 30, 40, 50]
print(numbers[1:4])    # [20, 30, 40]
print(numbers[::-1])   # reversed
```

Lesson 4.2 — Dictionaries in depth

```
phone = {
    "model": "Galaxy S26",
    "storage_gb": 512,
    "telephoto_camera": True
}
print(phone["model"])
phone["color"] = "Obsidian"
for key, value in phone.items():
    print(key, "->", value)
```

Lesson 4.3 — Looping over collections

```
scores = {"AI": 92, "Security": 88, "Apps": 75}
for subject, score in scores.items():
    if score >= 85:
        print(subject, "is a strength")
```

Exercises — Chapter 4

1. Build a contacts dictionary: store 3 friends, then print one, add one, delete one.
2. Write a program that counts how many times each word appears in a sentence (use a dictionary).
3. Combine two lists into a dictionary using `zip()`.

Chapter 5 — Functions

Lesson 5.1 — Defining and calling functions

```
def greet(name, greeting="Hello"):
    return f"{greeting}, {name}!"

print(greet("Alisha"))
print(greet("Omar", "Assalamu alaikum"))
```

Lesson 5.2 — Return values and scope

Variables created inside a function stay inside it (local scope). Use `return` to pass results out:

```
def discount(price, percent):
    return price * (1 - percent / 100)

final_price = discount(299, 15)
print(final_price)  # 254.15
```

Lesson 5.3 — Type hints and docstrings

```
def multiply(a: int, b: int) -> int:
    """Multiply two whole numbers."""
    return a * b
```

Exercises — Chapter 5

1. Write a function that checks if a number is prime.
2. Write a function that converts temperatures between Celsius and Fahrenheit.
3. Write a function that takes a list of prices and returns the total after a 10% discount.

Chapter 6 — Files

Lesson 6.1 — Reading files

```
with open("notes.txt", "r", encoding="utf-8") as f:
```

```
    content = f.read()
    print(content)
```

Lesson 6.2 — Writing files

```
with open("log.txt", "a", encoding="utf-8") as f:
    f.write("new entry at 9:00\n")
```

Lesson 6.3 — CSV files — the format of spreadsheets

```
import csv

with open("articles.csv", "r") as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row["title"], row["views"])
```

Exercises — Chapter 6

1. Read a text file and print how many lines it contains.
2. Append today's date to a journal file on every run.
3. Read a CSV of expenses and print the total.

Chapter 7 — Errors and Debugging

Lesson 7.1 — Reading tracebacks

Read a traceback **bottom-up**: the last line tells you the error type and message; the lines above show the call chain. `NameError` means a variable doesn't exist; `TypeError` means you used the wrong type; `IndexError` means a list index is out of range.

Lesson 7.2 — try/except

```
try:
    number = int(input("Enter a number: "))
    print(100 / number)
except ValueError:
    print("That isn't a number!")
except ZeroDivisionError:
    print("Can't divide by zero.")
```

Lesson 7.3 — Debugging techniques

- Insert `print()` statements to inspect variable values ("print debugging").
- `print(type(variable))` to check unexpected types.
- Comment out half the code to isolate the failing part (binary search your bug).

Exercises — Chapter 7

1. Write a program that keeps asking for a number until a valid one is entered (catch the error).
2. Deliberately create five different errors in one program, read each traceback, and explain them in comments.
3. Fix the debugging challenge:

```
def get_average(nums):
    total = 0
    for n in nums:
        total = n
    return total / len(nums)
```

Chapter 8 — Object-Oriented Programming

Lesson 8.1 — Classes and objects

```
class Gadget:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def describe(self):
        return f"{self.name} — ${self.price}"

phone = Gadget("Pixel", 799)
print(phone.describe())
```

Lesson 8.2 — Inheritance

```
class Smartphone(Gadget):
    def __init__(self, name, price, os):
        super().__init__(name, price)
        self.os = os

    def boot(self):
```

```
        return f"{self.name} booting {self.os}..."
```

Exercises — Chapter 8

1. Create a `Student` class with name, grade, and a method to print a report.
2. Create a `Laptop(Gadget)` subclass with a `battery_hours` attribute.
3. Build a small library system: `Book` class, `Library` class that stores books and can list/search.

Chapter 9 — Modules and the Standard Library

Lesson 9.1 — Importing modules

```
import random
import math
from datetime import datetime

print(random.randint(1, 6))
print(math.sqrt(144))
print(datetime.now())
```

Lesson 9.2 — Writing your own module

Save `helper.py`:

```
def triple(x):
    return x * 3
```

Then in another file:

```
from helper import triple
print(triple(7))  # 21
```

Lesson 9.3 — Must-know stdlib modules

Module	Use it for
`os`, `pathlib`	Files and folders

Module	Use it for
`json`	Reading/writing JSON
`csv`	Spreadsheet data
`requests`	Web APIs (install: `pip install requests`)
`datetime`	Dates and times
`random`	Randomness
`re`	Regular expressions

Exercises — Chapter 9

1. Write a password generator using `random`.
2. Parse a JSON file that stores a list of apps and print each name.
3. Create your own module with two useful functions and import it in a main program.

Chapter 10 — APIs and JSON

Lesson 10.1 — What an API is

An API (Application Programming Interface) lets your program ask another service for data over the internet. Most modern APIs return **JSON**, which looks like a Python dictionary.

Lesson 10.2 — Fetching data with requests

```
import requests

response = requests.get("https://api.github.com/zen")
print(response.status_code)   # 200 = OK
print(response.text)
```

For JSON:

```
data = requests.get("https://api.github.com/users/octocat").json()
print(data["login"], data["public_repos"])
```

Lesson 10.3 — Real project: simple weather app

```
import requests

city = input("City: ")
url = f"https://wttr.in/{city}?format=3"
print(requests.get(url).text)
```

Exercises — Chapter 10

1. Fetch the first 5 items from a public JSON API of your choice and print their names.
2. Build a program that checks the status code of the top 10 sites you visit daily.
3. Build a currency converter that reads live exchange rates from a public API.

Chapter 11 — Testing Your Code

Lesson 11.1 — assert and simple tests

```
def add(a, b):
    return a + b

assert add(2, 2) == 4
assert add(-1, 1) == 0
print("All tests passed")
```

Lesson 11.2 — pytest basics

Install with `pip install pytest`, then write tests in a file starting with `test_`:

```
def test_add():
    assert add(2, 2) == 4
```

Run with `pytest` in the terminal. Green dots = passing.

Exercises — Chapter 11

1. Write five assertions for a function of your choice.
2. Convert the Fibonacci function to tests with pytest.
3. Write a failing test, run it, then fix the code until it passes.

Chapter 12 — Capstone Projects

Project 1 — To-do list with file storage

Requirements: add, list, mark-done, delete tasks; save to a JSON file on every change; load on start. This project exercises everything from chapters 1–9.

Project 2 — Personal budget tracker

Read expenses from a CSV, group them by category, print totals, and save a summary. Exercises data handling and functions.

Project 3 — Flashcard quiz game

Load Q&A pairs from a file, quiz the user, track score, and offer a review round for missed cards. Exercises dictionaries, control flow and files.

Push all three to GitHub — the portfolio matters as much as the code.

Course glossary

Term	Meaning
Variable	A named container for a value
String	Text data, quoted with `"` or `'`
List	Ordered, changeable collection
Dictionary	Key-value collection
Function	Reusable named block of code
Loop	Repeats code until a condition
Exception	An error Python can catch
Class	Template for creating objects
Module	A file of reusable Python code
API	Interface for programs to talk over the web
JSON	Text format for structured data

Course complete! You now have a working foundation in Python: syntax, data structures, functions, files, errors, OOP, modules, APIs and testing — plus three portfolio projects. Keep going by building something for a real problem in your world. Next steps: pick a specialization (web, data, AI, security) and find our related articles on TechSparking to continue the journey.