Python Zero to Hero — A Beginner's Guide

A free guide by Parth Shah — ajconsultation.com
One-on-one tutoring in St. Augustine, Florida, or online.

Overview

This guide walks you through Python from the very beginning — no experience required. By the end of the 8 weeks, you will have written real code, solved practical problems, and built a small project you can show to others.

Python is one of the best first languages to learn. It reads almost like English, the syntax is clean, and you can see results quickly. This guide is built for self-study but works just as well alongside tutoring sessions.

Who This Guide Is For

- High school students exploring programming for the first time
- Adults changing careers into tech or data roles
- Summer learners who want a productive break
- Anyone who has tried to learn Python before but got stuck

You do not need a computer science background. You need a laptop, an internet connection, and a willingness to practice.

Why Python

Python is used in web development, data science, machine learning, automation, and more. It is consistently one of the top three most in-demand languages on job boards. More importantly for beginners: the feedback loop is fast. You write a line, you run it, you see what happens. That makes learning much more concrete.

The 8-Week Curriculum

Week 1 — Setup, Output, Variables, and Data Types

Goal: Get your environment running and write your first real programs.

Topics:

- Installing Python and VS Code (or using an online editor like Replit)

- Running your first script
- `print()` — how to display output
- Variables: what they are and how to name them
- Core data types: `int`, `float`, `str`, `bool`
- Type checking with `type()`
- Basic string operations: concatenation, f-strings

Practice Exercises:

1. Print your name, age, and favorite food using variables
2. Calculate the area of a rectangle using two variables
3. Write a program that prints a short introduction about yourself using an f-string
4. Experiment: what happens when you add a string and an integer?

Week 2 — Conditionals and Loops

Goal: Make your programs respond to different situations and repeat actions.

Topics:

- `if`, `elif`, `else` — making decisions
- Comparison operators: `==`, `!=`, `>`, `<`, `>=`, `<=`
- Logical operators: `and`, `or`, `not`
- `while` loops — repeat until a condition is false
- `for` loops — iterate over a range or sequence
- `range()` function
- `break` and `continue`

Practice Exercises:

1. Write a program that checks if a number is even or odd
2. Print the numbers 1 to 100, but skip multiples of 3
3. Write a simple guessing game: the user guesses a number between 1 and 10, and the program says "too high," "too low," or "correct"
4. Loop through the numbers 1 to 20 and print only the ones divisible by both 2 and 5

Week 3 — Functions

Goal: Write reusable blocks of code that do one thing well.

Topics:

- Defining a function with `def`
- Parameters and arguments
- Return values
- Default parameter values
- Scope: local vs. global variables
- Why functions matter (avoid repetition, easier to test)

Practice Exercises:

1. Write a function `greet(name)` that returns "Hello, [name]!"
2. Write a function `is_even(n)` that returns `True` or `False`
3. Write a function `celsius_to_fahrenheit(c)` that converts temperatures
4. Write a calculator function that takes two numbers and an operator (`+`, `-`, `*`, `/`) and returns the result
5. Refactor your guessing game from Week 2 to use functions

Week 4 — Lists and Dictionaries

Goal: Work with collections of data.

Topics:

- Lists: creating, indexing, slicing
- Common list methods: `append()`, `remove()`, `sort()`, `len()`
- Looping through lists
- List comprehensions (intro)
- Dictionaries: key-value pairs
- Common dict methods: `get()`, `keys()`, `values()`, `items()`
- Nested structures (a list of dictionaries)

Practice Exercises:

1. Create a list of your five favorite movies. Print each one on its own line using a loop.

2. Write a function that takes a list of numbers and returns the average
3. Create a dictionary representing a person (name, age, city). Print each field in a formatted sentence.
4. Write a program that counts how many times each word appears in a sentence (use a dictionary)
5. Create a list of student dictionaries, each with a name and grade. Print only the students who passed (grade >= 60).

Week 5 — File I/O and Exception Handling

Goal: Read from and write to files, and handle errors gracefully.

Topics:

- Opening files with `open()`
- Reading: `read()`, `readlines()`, iteration
- Writing and appending to files
- The `with` statement (context managers)
- What exceptions are and why they happen
- `try`, `except`, `finally`
- Common exceptions: `FileNotFoundError`, `ValueError`, `ZeroDivisionError`

Practice Exercises:

1. Write a program that saves a user's name and favorite color to a text file
2. Read that file back and print its contents
3. Write a program that reads a list of numbers from a file and prints their sum — handle the case where the file doesn't exist
4. Wrap your calculator function in a `try/except` block that catches division by zero and invalid input

---

Week 6 — Object-Oriented Programming Basics

Goal: Understand how to model real-world things as objects in code.

Topics:

- What OOP is and why it's used
- Classes and objects
- The `__init__` method

- Instance variables and methods
- `self` — what it means
- Creating multiple objects from one class
- String representation with `__str__`

Practice Exercises:

1. Create a `Dog` class with attributes `name` and `breed`, and a method `bark()` that prints "[name] says: Woof!"
2. Create a `BankAccount` class with a balance, and methods `deposit()`, `withdraw()`, and `get_balance()`
3. Create a `Student` class with name, grade, and a method `is_passing()` that returns `True` if grade >= 60
4. Instantiate three students from Week 4's list using your new class

---

Week 7 — Working with Libraries

Goal: Use Python's built-in and third-party libraries to do real tasks.

Topics:

- What a library/module is
- Importing: `import`, `from ... import`
- `os` module: file paths, directory listing, `os.path`
- `json` module: reading and writing JSON data
- `requests` library: making HTTP requests, reading API responses
- Brief intro to `pip` and installing packages

Practice Exercises:

1. Use `os` to list all files in a folder on your computer
2. Write a program that saves a dictionary to a JSON file and reads it back
3. Use the `requests` library to fetch data from a free public API (try `https://api.agify.io/?name=michael`) and print the result
4. Combine `requests` and `json` to pretty-print an API response

Week 8 — Mini Project Week

Goal: Apply everything you have learned to build one complete program.

Suggested Projects (pick one):

- **Personal Budget Tracker:** Users can log expenses by category, view totals, and save/load data from a file
- **Contact Book:** Store, search, and delete contacts saved to a JSON file, with a simple command-line menu
- **Quiz App:** Load questions from a file, quiz the user, track their score, and save results
- **Weather CLI Tool:** Use a weather API to let the user look up current conditions for any city

What Your Project Should Include:

- At least two functions
- A class or dictionary-based data structure
- File I/O (save and load data)
- Basic error handling
- A clear, working command-line interface

Steps:

1. Write out what your program will do in plain English before writing any code
2. Break it into small pieces and build one at a time
3. Test each piece as you go
4. Comment your code so someone else could follow it

Resources

Official and Core

- Python Official Docs — the authoritative reference
- Python Tutorial (official) — good supplement to this guide

Practice Platforms

- HackerRank — Python — beginner-friendly challenges, great for building confidence
- LeetCode — Easy problems — once you are comfortable with Week 1-4 material
- Exercism.io — free, mentor-reviewed exercises

- Replit — code in your browser without any setup

Supplemental Reading

- *Automate the Boring Stuff with Python* by Al Sweigart — free online at automatetheboringstuff.com — excellent practical examples

---

A Note on Getting Stuck

Getting stuck is normal. Every developer, including senior ones, spends a lot of time debugging. When you hit a wall:

1. Read the error message carefully — it usually tells you where the problem is
2. Search the exact error message on Google
3. Check the official docs
4. Take a break and come back fresh

If you are working with a tutor, bring your broken code to the session — debugging together is one of the fastest ways to learn.

Written by Parth Shah. For tutoring inquiries, visit ajconsultation.com.