

GUIDE • PROGRAMMING

Python from Zero



A Hands-On Beginner's Guide to Programming

WRITTEN BY

Your Name

EDITION

First edition, 2026

FORMAT

Editable guide

Python from Zero

A Hands-On Beginner's Guide to Programming

© 2026 Your Name. All rights reserved. This template is provided by Morphiq; replace the author and publisher details with your own.

This book is for general information and education. It is not professional, financial, medical or legal advice; consult a qualified professional for decisions about your own situation.

Examples, names, organisations and figures in this book are illustrative and should not be read as statistics.

Edition: First · Published by Morphiq Guides

How to Use This Book

Type every example yourself instead of copying it — your fingers learn as much as your eyes. Each chapter ends with exercises that build on the previous ones.

You need a computer with Python 3 installed. All examples work on Windows, macOS and Linux.

Inside every chapter

- A short list of what you will learn
- Clear explanations with examples and tables
- “Try this” activities to apply ideas straight away
- Key takeaways and practice questions

Contents

Getting Started with Python.....	6
Installing Python.....	6
Your First Program.....	6
Variables.....	6
Numbers, Strings and Input.....	8
Working with Numbers.....	8
Working with Text.....	8
Reading Input.....	8
Making Decisions with if.....	10
Conditions.....	10
Branching.....	10
Combining Conditions.....	10
Loops: Repeating Work.....	12
for Loops.....	12
while Loops.....	12
Controlling Loops.....	12
Lists and Tuples.....	13
Creating Lists.....	13
Changing Lists.....	13
Tuples.....	13
Dictionaries and Sets.....	14
Key–Value Pairs.....	14
Looping.....	14
Sets.....	14
Functions.....	15
Defining Functions.....	15
Returning Values.....	15
Default Arguments.....	15
Files and Errors.....	16
Reading Files.....	16
Writing Files.....	16
Handling Errors.....	16

Modules and Libraries.....	17
Importing.....	17
Standard Library Highlights.....	17
Installing Packages.....	17
Your First Project.....	19
Plan First.....	19
Build in Steps.....	19
Where to Go Next.....	19
Glossary and Resources.....	21
Glossary.....	21
Further Reading and Resources.....	21
About the Author.....	21

Getting Started with Python

Python is a friendly, readable programming language used for websites, data analysis, automation and artificial intelligence. In this chapter you will install Python and write your first program.

In this chapter you will learn

- How to install Python and an editor
- How to run a program
- What variables are

Installing Python

Download Python 3 from the official website and install it, ticking the option to add Python to your system path. Install a simple editor such as VS Code or use IDLE, which comes with Python.

Your First Program

Create a file called hello.py, type the line below and run it. Python prints the text in the terminal.

Variables

A variable is a name that refers to a value. Python works out the type automatically, so you can store numbers, text and more without declaring types.

- Names should describe the value
- Use lowercase with underscores
- Avoid starting names with numbers

Example

```
name = "Asha"  
age = 19  
print("Hello,", name)  
print("Next year you will be", age + 1)
```

Try this

Change the message to greet yourself by name and run the program again.

Key Takeaways

- Python is installed once and run from a terminal or editor.
- `print()` shows output.
- Variables store values with meaningful names.

Practice Questions

1. Install Python and run hello.py.
2. Create variables for your name and age and print them.
3. What happens if you misspell print?

Numbers, Strings and Input

Programs become useful when they work with data from the user. This chapter covers arithmetic, text and reading input.

In this chapter you will learn

- Arithmetic operators
- String operations
- Reading and converting input

Working with Numbers

Python supports + - * / for arithmetic, // for whole-number division, % for remainder and ** for powers.

Working with Text

Strings can be joined, repeated and sliced. Methods such as upper(), lower() and strip() transform text.

Reading Input

input() always returns text. Convert it with int() or float() before doing arithmetic.

Example

```
price = float(input("Price: "))
total = price * 1.18
print(f"Total with GST: {total:.2f}")
```

Operator	Meaning	Example	Result
+	Addition	7 + 2	9
//	Floor division	7 // 2	3
%	Remainder	7 % 2	1
**	Power	2 ** 5	32

Table 2: Common arithmetic operators

Try this

Write a program that asks for the price of an item and prints the price with 18% GST added.

Key Takeaways

- Use the right operator for the job.
- Strings have many helpful methods.
- Convert input before calculating.

Practice Questions

4. Write a program that converts Celsius to Fahrenheit.
5. Print your name in capital letters.
6. What does `17 % 5` return?

Making Decisions with if

Programs often need to choose what to do. The if statement runs code only when a condition is true.

In this chapter you will learn

- Comparison operators
- if, elif and else
- Combining conditions

Conditions

Comparisons such as ==, !=, <, > return True or False. Remember that = assigns a value while == compares.

Branching

Use elif for additional conditions and else for everything that remains. Indentation shows which lines belong to each branch.

Combining Conditions

Use and, or and not to build more complex conditions.

Example

```
marks = int(input("Marks: "))
if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
else:
    print("Keep practising")
```

Try this

Write a program that tells the user whether a number is even or odd.

Key Takeaways

- Conditions evaluate to True or False.
- Indentation defines blocks.
- and/or/not combine conditions.

Practice Questions

7. Write a program that prints the grade for a percentage.
8. Check if a year is a leap year.

9. What is the difference between = and ==?

Loops: Repeating Work

Loops let a program repeat actions without copying code. Python has for loops for sequences and while loops for repeating until a condition changes.

In this chapter you will learn

- for loops with range()
- while loops
- break and continue

for Loops

A for loop goes through each item in a sequence. range(1, 11) produces the numbers 1 to 10.

while Loops

A while loop repeats as long as its condition is true. Make sure something inside the loop eventually makes the condition false.

Controlling Loops

break exits a loop early; continue skips to the next round.

Example

```
total = 0
for n in range(1, 101):
    total += n
print("Sum:", total)
```

Try this

Print all numbers from 1 to 50 that are divisible by 3 or 5.

Key Takeaways

- for loops suit known sequences.
- while loops suit unknown repetitions.
- Avoid infinite loops.

Practice Questions

10. Print the multiplication table of 7.
11. Sum the numbers from 1 to 100.
12. Write a guessing game using while.

Lists and Tuples

Lists store collections of items in order. They are one of the most useful tools in Python.

In this chapter you will learn

- Creating and indexing lists
- Adding, removing and sorting
- Tuples and when to use them

Creating Lists

Lists use square brackets. Items are numbered from 0, and negative indexes count from the end.

Changing Lists

`append()` adds an item, `remove()` deletes one, `sort()` arranges items and `len()` counts them.

Tuples

Tuples are like lists but cannot be changed after creation, which makes them suitable for fixed data such as coordinates.

Example

```
marks = [78, 91, 66, 85]
marks.append(72)
marks.sort()
print(marks, "average:", sum(marks) / len(marks))
```

Try this

Create a shopping list, add two items, remove one and print it sorted.

Key Takeaways

- Lists are ordered and changeable.
- Indexes start at 0.
- Tuples are fixed.

Practice Questions

13. Store five marks in a list and print the average.
14. Find the largest item without `max()`.
15. When would you use a tuple?

Dictionaries and Sets

Dictionaries store data as key–value pairs, which makes looking things up fast and readable.

In this chapter you will learn

- Creating dictionaries
- Looping through keys and values
- Using sets to remove duplicates

Key–Value Pairs

A dictionary maps keys to values, such as a roll number to a student name.

Looping

`items()` returns pairs you can loop through. `get()` returns a default if a key is missing.

Sets

Sets hold unique items and support union and intersection.

Example

```
text = "to be or not to be"
count = {}
for w in text.split():
    count[w] = count.get(w, 0) + 1
print(count)
```

Try this

Build a dictionary of five countries and their capitals, then print them neatly.

Key Takeaways

- Dictionaries map keys to values.
- `get()` avoids errors for missing keys.
- Sets remove duplicates.

Practice Questions

16. Count how often each word appears in a sentence.
17. Store phone numbers by name.
18. Find common items in two lists.

Functions

Functions package code into reusable blocks with a name, inputs and an output.

In this chapter you will learn

- Defining and calling functions
- Parameters and return values
- Default arguments

Defining Functions

Use `def`, a name and parentheses. The indented body runs each time the function is called.

Returning Values

`return` sends a value back to the caller, so the result can be stored or used in calculations.

Default Arguments

Parameters can have default values, making functions flexible.

Example

```
def with_gst(price, rate=18):  
    return price * (1 + rate / 100)  
  
print(with_gst(500))  
print(with_gst(500, 5))
```

Try this

Rewrite your GST program as a function that takes price and rate.

Key Takeaways

- Functions reduce repetition.
- `return` gives back a result.
- Defaults make functions flexible.

Practice Questions

19. Write a function that returns the area of a circle.
20. Write a function that checks if a number is prime.
21. Why are functions useful?

Files and Errors

Real programs read and write files and must handle unexpected situations gracefully.

In this chapter you will learn

- Reading and writing text files
- Using with blocks
- Handling errors with try/except

Reading Files

open() returns a file object. A with block closes the file automatically.

Writing Files

Open a file with mode "w" to write or "a" to append.

Handling Errors

try/except catches errors such as a missing file or invalid input, so your program can respond helpfully instead of crashing.

Example

```
try:
    with open("notes.txt") as f:
        print(f.read())
except FileNotFoundError:
    print("No notes yet – start writing!")
```

Try this

Write a diary program that appends today's note to diary.txt.

Key Takeaways

- with blocks close files safely.
- Choose the right file mode.
- Catch expected errors.

Practice Questions

22. Write a program that saves notes to a file.
23. Count the lines in a file.
24. Handle non-numeric input gracefully.

Modules and Libraries

Python's strength comes from its standard library and thousands of free packages.

In this chapter you will learn

- Importing modules
- Useful standard modules
- Installing packages with pip

Importing

`import math` gives access to functions such as `math.sqrt()`. You can import specific names with `from ... import ...`

Standard Library Highlights

`random`, `datetime`, `os` and `csv` cover many everyday tasks.

- `random` — random choices and numbers
- `datetime` — dates and times
- `csv` — spreadsheets in text form

Installing Packages

`pip install package-name` downloads a package. Use virtual environments to keep projects separate.

Example

```
import random, datetime
print("Today:", datetime.date.today())
print("Dice:", random.randint(1, 6))
```

Try this

Write a program that picks a random quote from a list each time it runs.

Key Takeaways

- Modules extend Python.
- The standard library is large.
- `pip` installs extra packages.

Practice Questions

25. Print today's date using `datetime`.
26. Simulate a dice roll.

27. What is a virtual environment?

Your First Project

Bringing everything together, you will build a small expense tracker that stores entries in a CSV file and prints a monthly summary.

In this chapter you will learn

- Planning a small project
- Combining functions, files and dictionaries
- Next steps in your learning

Plan First

Write down what the program should do: add an expense, list expenses, show totals by category.

Build in Steps

Write one function at a time and test it before moving on. Save data to a CSV file so it persists between runs.

Where to Go Next

Explore web development, data analysis or automation depending on your interests. Build small projects regularly.

Example

```
import csv
def add(date, category, amount):
    with open("expenses.csv", "a", newline="") as f:
        csv.writer(f).writerow([date, category, amount])

def summary():
    totals = {}
    with open("expenses.csv") as f:
        for d, cat, amt in csv.reader(f):
            totals[cat] = totals.get(cat, 0) + float(amt)
    return totals
```

Try this

Extend the tracker to show a summary for a chosen month.

Key Takeaways

- Plan before coding.
- Build and test step by step.
- Keep learning through projects.

Practice Questions

- 28. Add a feature to delete an expense.
- 29. Show the category with the highest spending.
- 30. What project will you build next?

Glossary and Resources

Glossary

Term	Meaning
Variable	A name that refers to a value
Function	A reusable block of code
Loop	Code that repeats
List	An ordered, changeable collection
Dictionary	A collection of key–value pairs
Module	A file of Python code you can import
Exception	An error detected while the program runs

Further Reading and Resources

- The official Python documentation and tutorial
- A code editor such as VS Code
- Practice websites with beginner exercises
- Your college or school programming club

About the Author

Write a short note about your programming experience and who this guide is for.

Your notes

Use this space to write the three ideas from this book you will act on first, and the date you will start.