LEARNHUB / TECHNICAL COURSE LIBRARY

Python Programming

Course introduction

Build a strong Python foundation by learning to turn small ideas into readable programs, then shape those skills into beginner-friendly projects.



Level: Beginner · Category: Programming

Original, beginner-friendly study content designed to accompany the course syllabus and its linked video lesson.

Contents

- How to use this resource
- Learning map
- Chapter 01 · Python foundations and values
- Chapter 02 · Conditions and decisions
- Chapter 03 · Loops and collections
- Chapter 04 · Functions and reusable design
- Chapter 05 · Objects and starter projects
- Summary and next steps

Course purpose: Build a strong Python foundation by learning to turn small ideas into readable programs, then shape those skills into beginner-friendly projects.

How to use this resource

Read the chapter titles first, then study the explanations and examples. Pause at each practice prompt before reading its suggested answer. Write down questions that remain and revisit them during revision.

Learning map



Chapter 01 · Python foundations and values

The central idea

A Python program is a sequence of statements evaluated by the interpreter. A variable name refers to a value such as an integer, string, Boolean, or collection.

How to think about it

Start with expressions, assignment, input, and output. Python's indentation groups statements into blocks. Choose names that describe data, and use built-in types before inventing complex structures.

First example

A temperature converter reads a Celsius value, computes `(celsius * 9 / 5) + 32`, and prints a labeled Fahrenheit result.

Try this

Write a short script that asks for a name and a temperature, then prints a friendly message with the converted value.

A way to check your answer

> Use `input()` for text, convert the numeric input with `float()`, calculate the Fahrenheit value, then format it in an f-string. Keep the name as a string.

Chapter 02 · Conditions and decisions

The central idea

A conditional selects a block of code based on a Boolean expression. `if`, `elif`, and `else` make mutually exclusive paths explicit.

How to think about it

Compare values with relational operators and combine tests with `and`, `or`, and `not`. Order branches from the most specific case to the general case. Test boundary values, not just typical inputs.

First example

A shipping calculator applies a free-shipping rule when an order total reaches a threshold; otherwise it adds a delivery fee.

Try this

Classify a score as below pass, pass, or distinction, and state how a score exactly on each boundary should behave.

A way to check your answer

Use an `if` branch for scores below the pass boundary, an `elif` branch for the pass range, and `else` for the remaining range. Confirm whether boundaries use `<` or `<=`.

Chapter 03 · Loops and collections

The central idea

A loop repeats a block. `for` visits items in an iterable; `while` repeats while a condition remains true. Lists store ordered, changeable collections of values.

How to think about it

Prefer `for` when the amount of work is tied to a collection or range. Use `while` for a condition-controlled process and ensure that each iteration can make progress toward stopping. Accumulators summarize values as they are visited.

First example

A receipt totals a list of prices by adding one price to a running total during each loop iteration.

Try this

Given a list of quiz scores, calculate the count and average of scores above a chosen threshold.

A way to check your answer

> Initialize a count and sum to zero; visit each score; update both when it qualifies. Divide by the count only after checking that the count is not zero.

Chapter 04 · Functions and reusable design

The central idea

A function is a named unit of behavior that accepts inputs, performs a focused task, and may return a result.

How to think about it

Use parameters to make a function reusable and `return` to send a value back to its caller. Keep input/output at the program boundary when possible so logic can be tested without manual interaction.

First example

A `calculate_total(prices, tax_rate)` function returns the subtotal plus tax; a separate display function formats the result for a receipt.

Try this

Refactor a repeated discount calculation into a function and describe its input, output, and one edge case.

A way to check your answer

> Accept the original price and discount rate as parameters, return the computed price, and test a zero discount and a rate outside the allowed range.

Chapter 05 · Objects and starter projects

The central idea

A class describes a kind of object. An object combines state stored in attributes with behavior implemented by methods.

How to think about it

Use classes when data and the operations that maintain it belong together. A constructor initializes valid state. Begin with small projects that have clear inputs, state changes, and outputs before adding interfaces or external services.

First example

A `BankAccount` class stores a balance and exposes deposit and withdraw methods that reject invalid amounts.

Try this

Sketch a small quiz application: identify one class, its state, two methods, and one rule that prevents invalid state.

A way to check your answer

A `Question` can hold a prompt, choices, and correct answer; a method can check a submitted choice. Validate that the answer is one of the offered choices.

Summary and next steps

Python Programming brings together 5 connected topics. Review each definition, explain one example without looking, and then try the practice prompt again.

Key ideas to carry forward

Python foundations and values: A Python program is a sequence of statements evaluated by the interpreter. A variable name refers to a value such as an integer, string, Boolean, or collection.

Conditions and decisions: A conditional selects a block of code based on a Boolean expression. `if`, `elif`, and `else` make mutually exclusive paths explicit.

Loops and collections: A loop repeats a block. `for` visits items in an iterable; `while` repeats while a condition remains true. Lists store ordered, changeable collections of values.

Functions and reusable design: A function is a named unit of behavior that accepts inputs, performs a focused task, and may return a result.

Objects and starter projects: A class describes a kind of object. An object combines state stored in attributes with behavior implemented by methods.

A short review routine

1. Recall the main idea from each chapter before rereading. 2. Work one example by hand. 3. Explain a common mistake and how you would catch it. 4. Revisit a difficult topic after a short break.

Course outcomes

- Read and write Python expressions, variables, and basic data structures
- Use conditions and loops to control a program
- Organize reusable logic with functions and classes
- Plan and complete a small Python project

This course introduction is an original LearnHub learning aid. Use the referenced ideas for study, then consult the course video and the rest of the library for additional practice.