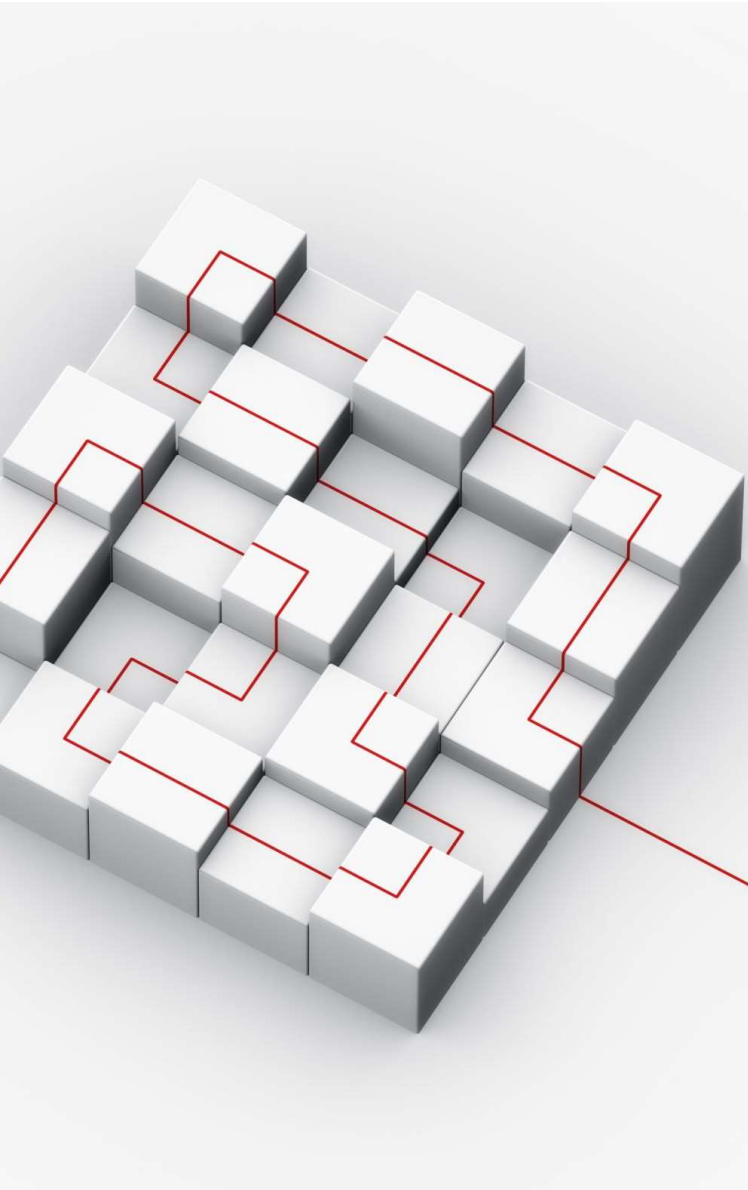


Module

Coding Basics



This chapter covers

- Module Creation
- Module Usage
- Module Namespaces
- Reloading Modules

Introduction

- Python modules are easy to create; they're just files of Python program code created with a text editor.
- You don't need to write special syntax to tell Python you're making a module; almost any text file will do.
- Because Python handles all the details of finding and loading modules, modules are also easy to use; clients simply import a module, or specific names a module defines, and use the objects they reference.

Activity

Module Creation

Activity

Module Usage

Using the LEGB Rule for Python Scope

- **Local** (or function) scope is the code block or body of any Python function or lambda expression. This Python scope contains the names that you define inside the function.
- **Enclosing** (or nonlocal) scope is a special scope that only exists for nested functions. If the local scope is an inner or nested function, then the enclosing scope is the scope of the outer or enclosing function.
- **Global** (or module) scope is the top-most scope in a Python program, script, or module.
- **Built-in** scope is a special Python scope that's created or loaded whenever you run a script or open an interactive session.

Module Namespaces

- Modules are probably best understood as simply packages of names - i.e., places to define names you want to make visible to the rest of a system.
- Technically, modules usually correspond to files, and Python creates a module object to contain all the names assigned in a module file.
- But in simple terms, modules are just namespaces (places where names are created), and the names that live in a module are called its *attributes*.

Reloading Modules

- As we've seen, a module's code is run only once per process by default.
- To force a module's code to be reloaded and rerun, you need to ask Python to do so explicitly by calling the reload built-in function.

The End