Developer Tools

Over the course of its lifetime, Python has evolved an extensive ecosystem of modules intended to make the lives of Python developers easier by eliminating the need to build everything from scratch. That same philosophy has been applied to the tools developers use to do their work, even if they are not used in the final version of a program. This chapter covers the modules included with Python to provide facilities for common development tasks such as testing, debugging, and profiling.

The most basic form of help for a developer is the documentation for code they are using. The `pydoc` module generates formatted reference documentation from the docstrings included in the source code for any importable module.

Python includes two testing frameworks for automatically exercising code and verifying that it works correctly. `doctest` extracts test scenarios from examples included in documentation, either inside the source or as stand-alone files. `unittest` is a full-featured automated testing framework with support for fixtures, pre-defined test suites, and test discovery.

The `trace` module monitors the way Python executes a program, producing a report showing how many times each line was run. That information can be used to find code paths that are not being tested by an automated test suite, and to study the function call graph to find dependencies between modules.

Writing and running tests will uncover problems in most programs. Python helps make debugging easier, since in most cases unhandled errors are printed to the console as tracebacks. When a program is not running in a text console environment, `traceback` can be used to prepare similar output for a log file or message dialog. For situations where a standard traceback does not provide enough information, use `cgitb` to see details like local variable settings at each level of the stack and source context. `cgitb` can also format tracebacks in HTML, for reporting errors in web applications.

Once the location of a problem is identified, stepping through the code using the interactive debugger in the `pdb` module can make it easier to fix by showing what path through the code was followed to get to the error situation, and experimenting with changes using live objects and code.

After a program is tested and debugged so that it works correctly, the next step is to work on performance. Using `profile` and `timeit`, a developer can measure the speed of a program and find the slow parts so they can be isolated and improved.

It is important to indent source code consistently in a langauge like Python, where white-space is significant. The `tabnanny` module provides a scanner to report on ambiguous use of indentation, and can be used in tests to ensure that code meets a minimum standard before it is checked in to the source repository.

Python programs are run by giving the interpreter a byte-compiled version of the original program source. The byte-compiled versions can be created on-the-fly, or once when the program is packaged. The `compileall` module exposes the interface used by installation programs and packaging tools to create files containing the byte code for a module. It can be used in a development environment to make sure a file does not have any syntax errors and to build the byte-compiled files to package when the program is released.

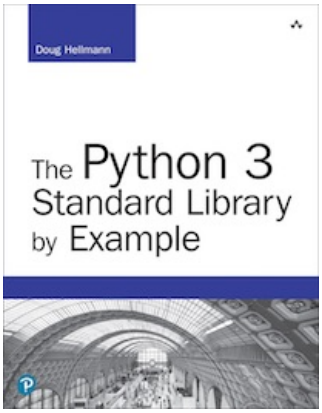
At the source code level, the `pyclbr` module provides a class browser that can be used by a text editor or other program to scan Python source for interesting symbols such as functions and classes, without importing the code and potentially triggering side-effects.

Python virtual environments, managed by `venv`, define isolated environments for installing packages and running programs. They make it easy to test the same program with different versions of dependencies, and to install different programs with conflicting dependencies on the same computer.

Taking advantage of the large ecosystem of extension modules, frameworks, and tools available through the Python Package Index requires a package installer. Python's package installer, pip, is not distributed with the interpreter, because of the long release cycle for the language compared to desired updates to the tool. The `ensurepip` module can be used to install the latest version of pip.

- pydoc — Online Help for Modules
- doctest — Testing Through Documentation
- unittest — Automated Testing Framework
- trace — Follow Program Flow
- traceback — Exceptions and Stack Traces
- cgitb — Detailed Traceback Reports
- pdb — Interactive Debugger
- profile and pstats — Performance Analysis
- timeit — Time the execution of small bits of Python code.
- tabnanny — Indentation validator
- compileall — Byte-compile Source Files
- pyclbr — Class Browser
- venv — Create Virtual Environments
- ensurepip — Install the Python Package Installer

This page was last updated 2016-12-31.



Get the book

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?