Data Persistence and Exchange

There are two aspects to preserving data for long-term use: converting the data back and forth between the object in-memory and the storage format, and working with the storage of the converted data. The standard library includes a variety of modules that handle both aspects in different situations.

Two modules convert objects into a format that can be transmitted or stored (a process known as *serializing*). It is most common to use `pickle` for persistence, since it is integrated with some of the other standard library modules that actually store the serialized data, such as `shelve`. `json` is more frequently used for web-based applications, however, since it integrates better with existing web service storage tools.

Once the in-memory object is converted to a format that can be saved, the next step is to decide how to store the data. A simple flat-file with serialized objects written one after the other works for data that does not need to be indexed in any way. Python includes a collection of modules for storing key-value pairs in a simple database using one of the DBM format variants when an indexed lookup is needed.

The most straightforward way to take advantage of the DBM format is `shelve`. Open the shelve file, and access it through a dictionary-like API. Objects saved to the database are automatically pickled and saved without any extra work by the caller.

One drawback of `shelve`, though, is that when using the default interface there is no way to predict which DBM format will be used, since it selects one based on the libraries available on the system where the database is created. The format does not matter if an application will not need to share the database files between hosts with different libraries, but if portability is a requirement, use one of the classes in the module to ensure a specific format is selected.

For web applications that work with data in JSON already, using `json` and `dbm` provides another persistence mechanism. Using `dbm` directly is a little more work than `shelve` because the DBM database keys and values must be strings, and the objects will not be re-created automatically when the value is accessed in the database.

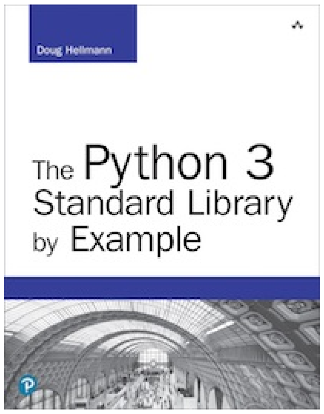
The `sqlite3` in-process relational database is available with most Python distributions for storing data in more complex arrangements than key/value pairs. It stores its database in memory or in a local file, and all access is from within the same process so there is no network communication lag. The compact nature of `sqlite3` makes it especially well suited for embedding in desktop applications or development versions of web apps.

There are also modules for parsing more formally defined formats, useful for exchanging data between Python programs and applications written in other languages. `xml.etree.ElementTree` can parse XML documents, and provides several operating modes for different applications. Besides the parsing tools, `ElementTree` includes an interface for creating well-formed XML documents from objects in memory. The `csv` module can read and write tabular data in formats produced by spreadsheets or database applications, making it useful for bulk loading data, or converting the data from one format to another.

io — Text, Binary, and Raw Stream I/O Tools   pickle — Object Serialization

This page was last updated 2016-12-03.

Navigation



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?

This Site

- Module Index
- Index

Other Writing

- Blog
- The Python Standard Library By Example