Chapter 25. Extending and Embedding Classic Python

The content of this chapter has been abbreviated for the print edition of this book. The full content is **available online**, *as described in* **"Online Material"**.

CPython runs on a portable, C-coded virtual machine. Python's built-in objects—such as numbers, sequences, dictionaries, sets, and files—are coded in C, as are several modules in Python's standard library. Modern platforms support dynamically loaded libraries, with file extensions such as *.dll* on Windows, *.so* on Linux, and *.dylib* on Mac: building Python produces such binary files. You can code your own extension modules for Python in C (or any language that can produce C-callable libraries), using the Python C API covered in this chapter. With this API, you can produce and deploy dynamic libraries that Python scripts and interactive sessions can later use with the `import` statement, covered in **"The import Statement"**.

Extending Python means building modules that Python code can `import` to access the features the modules supply. *Embedding* Python means executing Python code from an application coded in another language. For such execution to be useful, Python code must in turn be able to access some of your application's functionality. In practice, therefore, embedding implies some extending, as well as a few embedding-specific operations. The three main reasons for wishing to extend Python can be summarized as follows:

- Reimplementing some functionality (originally coded in Python) in a lower-level language, hoping to get better performance

- Letting Python code access some existing functionality supplied by libraries coded in (or, at any rate, callable from) lower-level languages
- Letting Python code access some existing functionality of an application that is in the process of embedding Python as the application's scripting language

Embedding and extending are covered in Python's online documentation; there, you can find an in-depth **tutorial** and an extensive **reference manual**. Many details are best studied in Python's extensively documented C sources. Download Python's source distribution and study the sources of Python's core, C-coded extension modules, and the example extensions supplied for this purpose.

Online Material

THIS CHAPTER ASSUMES SOME KNOWLEDGE OF C

Although we include some non-C extension options, to extend or embed Python using the C API you must know the C and/or C++ programming languages. We do not cover C and C++ in this book, but there are many print and online resources that you can consult to learn them. Most of the online content of this chapter assumes that you have at least some knowledge of C.

In the **online version of this chapter**, you will find the following sections:

"Extending Python with Python's C API"

Includes reference tables and examples for creating C-coded Python extension modules that you can import into your Python programs, showing how to code and build such modules. This section includes two complete examples:

- An extension implementing custom methods for manipulating `dicts`
- An extension defining a custom type

“Extending Python Without Python’s C API”

Discusses (or, at least, mentions and links to) several utilities and libraries that support creating Python extensions that do not directly require C or C++ programming,[1] including the third-party tools **F2PY**, **SIP**, **CLIF**, **`cppyy`**, **`pybind11`**, **Cython**, **CFFI**, and **HPy**, and standard library module **`ctypes`**. This section includes a complete example on how to create an extension using Cython.

“Embedding Python”

Includes reference tables and a conceptual overview of embedding a Python interpreter within a larger application, using Python’s C API for embedding.

[1] There are many other such tools, but we tried to pick just the most popular and promising ones.