Python experimental support for free threading

Release 3.13.4

Guido van Rossum and the Python development team

June 03, 2025

Python Software Foundation
Email: docs@python.org

Contents

1 Installation **2**

2 Identifying free-threaded Python **2**

3 The global interpreter lock in free-threaded Python **2**

4 Thread safety **2**

5 Known limitations **2**
5.1 Immortalization . . . 3
5.2 Frame objects . . . 3
5.3 Iterators . . . 3
5.4 Single-threaded performance . . . 3

Index **4**

Starting with the 3.13 release, CPython has experimental support for a build of Python called free threading where the global interpreter lock (GIL) is disabled. Free-threaded execution allows for full utilization of the available processing power by running threads in parallel on available CPU cores. While not all software will benefit from this automatically, programs designed with threading in mind will run faster on multi-core hardware.

The free-threaded mode is experimental and work is ongoing to improve it: expect some bugs and a substantial single-threaded performance hit.

This document describes the implications of free threading for Python code. See freethreading-extensions-howto for information on how to write C extensions that support the free-threaded build.

See also

PEP 703 – Making the Global Interpreter Lock Optional in CPython for an overall description of free-threaded Python.

1 Installation

Starting with Python 3.13, the official macOS and Windows installers optionally support installing free-threaded Python binaries. The installers are available at https://www.python.org/downloads/.

For information on other platforms, see the Installing a Free-Threaded Python, a community-maintained installation guide for installing free-threaded Python.

When building CPython from source, the `--disable-gil` configure option should be used to build a free-threaded Python interpreter.

2 Identifying free-threaded Python

To check if the current interpreter supports free-threading, `python -VV` and `sys.version` contain "experimental free-threading build". The new `sys._is_gil_enabled()` function can be used to check whether the GIL is actually disabled in the running process.

The `sysconfig.get_config_var("Py_GIL_DISABLED")` configuration variable can be used to determine whether the build supports free threading. If the variable is set to `1`, then the build supports free threading. This is the recommended mechanism for decisions related to the build configuration.

3 The global interpreter lock in free-threaded Python

Free-threaded builds of CPython support optionally running with the GIL enabled at runtime using the environment variable `PYTHON_GIL` or the command-line option `-X gil`.

The GIL may also automatically be enabled when importing a C-API extension module that is not explicitly marked as supporting free threading. A warning will be printed in this case.

In addition to individual package documentation, the following websites track the status of popular packages support for free threading:

- https://py-free-threading.github.io/tracking/
- https://hugovk.github.io/free-threaded-wheels/

4 Thread safety

The free-threaded build of CPython aims to provide similar thread-safety behavior at the Python level to the default GIL-enabled build. Built-in types like `dict`, `list`, and `set` use internal locks to protect against concurrent modifications in ways that behave similarly to the GIL. However, Python has not historically guaranteed specific behavior for concurrent modifications to these built-in types, so this should be treated as a description of the current implementation, not a guarantee of current or future behavior.

> **Note**
>
> It's recommended to use the `threading.Lock` or other synchronization primitives instead of relying on the internal locks of built-in types, when possible.

5 Known limitations

This section describes known limitations of the free-threaded CPython build.

5.1 Immortalization

The free-threaded build of the 3.13 release makes some objects immortal. Immortal objects are not deallocated and have reference counts that are never modified. This is done to avoid reference count contention that would prevent efficient multi-threaded scaling.

An object will be made immortal when a new thread is started for the first time after the main thread is running. The following objects are immortalized:

- function objects declared at the module level
- method descriptors
- code objects
- module objects and their dictionaries
- classes (type objects)

Because immortal objects are never deallocated, applications that create many objects of these types may see increased memory usage. This is expected to be addressed in the 3.14 release.

Additionally, numeric and string literals in the code as well as strings returned by `sys.intern()` are also immortalized. This behavior is expected to remain in the 3.14 free-threaded build.

5.2 Frame objects

It is not safe to access frame objects from other threads and doing so may cause your program to crash . This means that `sys._current_frames()` is generally not safe to use in a free-threaded build. Functions like `inspect.currentframe()` and `sys._getframe()` are generally safe as long as the resulting frame object is not passed to another thread.

5.3 Iterators

Sharing the same iterator object between multiple threads is generally not safe and threads may see duplicate or missing elements when iterating or crash the interpreter.

5.4 Single-threaded performance

The free-threaded build has additional overhead when executing Python code compared to the default GIL-enabled build. In 3.13, this overhead is about 40% on the pyperformance suite. Programs that spend most of their time in C extensions or I/O will see less of an impact. The largest impact is because the specializing adaptive interpreter (**PEP 659**) is disabled in the free-threaded build. We expect to re-enable it in a thread-safe way in the 3.14 release. This overhead is expected to be reduced in upcoming Python release. We are aiming for an overhead of 10% or less on the pyperformance suite compared to the default GIL-enabled build.

Index

E

environment variable
    PYTHON_GIL, 2

P

Python Enhancement Proposals
    PEP 659, 3
    PEP 703, 1
PYTHON_GIL, 2