`!atexit` --- Exit handlers



- !atexit Example

atexit

The `!atexit` module defines functions to register and unregister `exit handlers`: functions that are automatically executed "at exit", that is, upon normal program termination (for instance, if `sys.exit` is called or the main module's execution completes) or, more generally, upon `interpreter shutdown`.

At exit, all registered exit handlers are called in the *reverse* order in which they were registered. If you register `A`, `B`, and `C`, at interpreter shutdown time they will be run in the order `C`, `B`, `A`. The assumption is that lower level modules will normally be imported before higher level modules and thus must be cleaned up later.

If an exception is raised during execution of an exit handler, a traceback is printed (unless `SystemExit` is raised) and the exception information is saved. After all exit handlers have had a chance to run, the last exception to be raised is re-raised.

In programs that use multiple interpreters, each interpreter has its own stack of exit handlers, which are executed when the interpreter shuts down (for example, with `concurrent.interpreters.Interpreter.close` or the C API `Py_EndInterpreter`). Registration functions in this module only affect the interpreter they are called from.

Note: Exit handlers are not called when the program is killed by a signal not handled by Python, when a Python fatal internal error is detected, or when `os._exit` is called.

Note: The effect of registering or unregistering functions from within a cleanup function is undefined.

> [!WARNING] When writing exit handlers, especially in C API extensions, keep in mind that other exit handlers may still run arbitrary Python code after you clean up. Such code should succeed or fail with an exception, rather than crash.

3.12 Attempts to start a new thread or `os.fork` a new process in an exit handler now leads to `RuntimeError`. Previously, this could cause race conditions between the main Python runtime thread

freeing thread states while internal `threading` routines or the new process try to use that state, which could lead to crashes rather than clean shutdown.

3.7 When used with subinterpreters, registered functions are local to the interpreter they were registered in.

register(func, *args,*kwargs)

Register *func* as an exit handler. Any optional arguments that are to be passed to *func* must be passed as arguments to `register`. It is possible to register the same function and arguments more than once.

This function returns *func*, which makes it possible to use it as a decorator.

unregister(func)

Remove *func* from the list of exit handlers. `unregister` silently does nothing if *func* was not previously registered. If *func* has been registered more than once, every occurrence of that function in the `!atexit` call stack will be removed. Equality comparisons (`==`) are used internally during unregistration, so function references do not need to have matching identities.

Module `readline`
Useful example of `!atexit` to read and write `readline` history files.

`!atexit` Example

The following simple example demonstrates how a module can initialize a counter from a file when it is imported and save the counter's updated value automatically when the program terminates without relying on the application making an explicit call into this module at termination. :

```
try:
    with open('counterfile') as infile:
        _count = int(infile.read())
except FileNotFoundError:
    _count = 0

def incrcounter(n):
    global _count
    _count = _count + n

def savecounter():
```

```
    with open('counterfile', 'w') as outfile:
        outfile.write('%d' % _count)

import atexit

atexit.register(savecounter)
```

Positional and keyword arguments may also be passed to `register` to be passed along to the registered function when it is called:

```
def goodbye(name, adjective):
    print('Goodbye %s, it was %s to meet you.' % (name, adjective))

import atexit

atexit.register(goodbye, 'Donny', 'nice')
# or:
atexit.register(goodbye, adjective='nice', name='Donny')
```

Usage as a `decorator`:

```
import atexit

@atexit.register
def goodbye():
    print('You are now leaving the Python sector.')
```

This only works with functions that can be called without arguments.

□□□ #2
□ □□□ □□□ 2026-10-31 17:04:08 UTC
□ □□□ □□□ 2026-10-31 17:04:10 UTC