## 28.8. `atexit` — Exit handlers¶

New in version 2.0.

The `atexit` module defines a single function to register cleanup functions. Functions thus registered are automatically executed upon normal interpreter termination.

Note: the functions registered via this module are not called when the program is killed by a signal, when a Python fatal internal error is detected, or when `os._exit()` is called.

This is an alternate interface to the functionality provided by the `sys.exitfunc` variable.

Note: This module is unlikely to work correctly when used with other code that sets `sys.exitfunc`. In particular, other core Python modules are free to use `atexit` without the programmer's knowledge. Authors who use `sys.exitfunc` should convert their code to use `atexit` instead. The simplest way to convert code that sets `sys.exitfunc` is to import `atexit` and register the function that had been bound to `sys.exitfunc`.

`atexit.register`(*func*[, **args*[, ***kargs*]])¶

Register *func* as a function to be executed at termination. Any optional arguments that are to be passed to *func* must be passed as arguments to `register()`.

At normal program termination (for instance, if `sys.exit()` is called or the main module's execution completes), all functions registered are called in last in, first out order. 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 the exit handlers, 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.

Changed in version 2.6: This function now returns *func* which makes it possible to use it as a decorator without binding the original name to `None`.

See also

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

### 28.8.1. `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:
   _count = int(open("/tmp/counter").read())
except IOError:
   _count = 0

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

def savecounter():
   open("/tmp/counter", "w").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 obviously only works with functions that don't take arguments.


**Table Of Contents**

28.8. `atexit` — Exit handlers

- 28.8.1. `atexit` Example



**Previous topic**

28.7. `abc` — Abstract Base Classes

**Next topic**

28.9. `traceback` — Print or retrieve a stack traceback

**This Page**

- Show Source

**Navigation**

- index
- modules |
- next |
- previous |
- Python v2.6.4 documentation »
- The Python Standard Library »
- 28. Python Runtime Services »