# 29.8. `atexit` — Exit handlers

The `atexit` module defines functions to register and unregister cleanup functions. Functions thus registered are automatically executed upon normal interpreter termination. `atexit` runs these functions in the *reverse* order in which they were registered; if you register `A`, `B`, and `C`, at interpreter termination time they will be run in the order `C`, `B`, `A`.

**Note:** The functions registered via this module 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.

`atexit.`**`register`**(*func*, **args*, ***kwargs*)

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()`. It is possible to register the same function and arguments more than once.

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.

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

`atexit.`**`unregister`**(*func*)

Remove *func* from the list of functions to be run at interpreter shutdown. After calling `unregister()`, *func* is guaranteed not to be called when the interpreter shuts down, even if it was registered more than once. `unregister()` silently does nothing if *func* was not previously registered.

**See also:**

**Module** `readline`

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

## 29.8.1. `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:
    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.