# Exception Handling

sys includes features for trapping and working with exceptions.

## Unhandled Exceptions

Many applications are structured with a main loop that wraps execution in a global exception handler to trap errors not handled at a lower level. Another way to achieve the same thing is by setting the `sys.excepthook` to a function that takes three arguments (the error type, error value, and traceback) and let it deal with unhandled errors.

```
# sys_excepthook.py

import sys


def my_excepthook(type, value, traceback):
    print('Unhandled error:', type, value)


sys.excepthook = my_excepthook

print('Before exception')

raise RuntimeError('This is the error message')

print('After exception')
```

Since there is no `try:except` block around the line where the exception is raised the following call to `print()` is not run, even though the except hook is set.

```
$ python3 sys_excepthook.py

Before exception
Unhandled error: <class 'RuntimeError'> This is the error
message
```

## Current Exception

There are times when an explicit exception handler is preferred, either for code clarity or to avoid conflicts with libraries that try to install their own excepthook. In these cases, a common handler function can be created that does not need to have the exception object passed to it explicitly by calling `exc_info()` to retrieve the current exception for a thread.

The return value of `exc_info()` is a three member tuple containing the exception class, an exception instance, and a traceback. Using `exc_info()` is preferred over the old form (with `exc_type`, `exc_value`, and `exc_traceback`) because it is thread-safe.

```
# sys_exc_info.py

import sys
import threading
import time


def do_something_with_exception():
    exc_type, exc_value = sys.exc_info()[:2]
    print('Handling {} exception with message "{}" in {}'.format(
        exc_type.__name__, exc_value,
        threading.current_thread().name))
```

```
def cause_exception(delay):
    time.sleep(delay)
    raise RuntimeError('This is the error message')


def thread_target(delay):
    try:
        cause_exception(delay)
    except RuntimeError:
        do_something_with_exception()


threads = [
    threading.Thread(target=thread_target, args=(0.3,)),
    threading.Thread(target=thread_target, args=(0.1,)),
]

for t in threads:
    t.start()
for t in threads:
    t.join()
```

This example avoids introducing a circular reference between the traceback object and a local variable in the current frame by ignoring that part of the return value from `exc_info()`. If the traceback is needed (for example, so it can be logged), explicitly delete the local variable (using `del`) to avoid cycles.

```
$ python3 sys_exc_info.py

Handling RuntimeError exception with message "This is the error
message" in Thread-2
Handling RuntimeError exception with message "This is the error
message" in Thread-1
```

## Previous Interactive Exception

In the interactive interpreter, there is only one thread of interaction. Unhandled exceptions in that thread are saved to three variables in sys (`last_type`, `last_value`, and `last_traceback`) to make it easy to retrieve them for debugging. Using the postmortem debugger in pdb avoids any need to use the values directly.

```
$ python3
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  5 2014, 20:42:22)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> def cause_exception():
...     raise RuntimeError('This is the error message')
...
>>> cause_exception()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in cause_exception
RuntimeError: This is the error message
>>> import pdb
>>> pdb.pm()
> <stdin>(2)cause_exception()
(Pdb) where
  <stdin>(1)<module>()
> <stdin>(2)cause_exception()
(Pdb)
```

### See also

- exceptions – Built-in errors
- pdb – Python debugger
- traceback – Module for working with tracebacks

Memory Management and Limits

Low-level Thread Support

**Quick Links**

Unhandled Exceptions
Current Exception
Previous Interactive Exception

*This page was last updated 2016-12-18.*

**Navigation**

Memory Management and Limits
Low-level Thread Support



*The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.*

*Looking for examples for Python 2?*

**This Site**

Module Index
Index



**Other Writing**

Blog
The Python Standard Library By Example