

warnings — Non-fatal Alerts

Purpose: Deliver non-fatal alerts to the user about issues encountered when running a program.

The warnings module was introduced by [PEP 230](#) as a way to warn programmers about changes in language or library features in anticipation of backwards incompatible changes coming with Python 3.0. It can also be used to report recoverable configuration errors or feature degradation from missing libraries. It is better to deliver user-facing messages via the [logging](#) module, though, because warnings sent to the console may be lost.

Since warnings are not fatal, a program may encounter the same warn-able situation many times in the course of running. The warnings module suppresses repeated messages from the same source to cut down on the annoyance of seeing the same warning over and over. The output can be controlled on a case-by-case basis, using the command line options to the interpreter or by calling functions found in warnings.

Categories and Filtering

Warnings are categorized using subclasses of the built-in exception class `Warning`. Several standard values are described in the online documentation for the exceptions module, and custom warnings can be added by subclassing from `Warning`.

Warnings are processed based on *filter* settings. A filter consists of five parts: the action, message, category, module, and line number. The message portion of the filter is a regular expression that is used to match the warning text. The category is a name of an exception class. The module contains a regular expression to be matched against the module name generating the warning. And the line number can be used to change the handling on specific occurrences of a warning.

When a warning is generated, it is compared against all of the registered filters. The first filter that matches controls the action taken for the warning. If no filter matches, the default action is taken. The actions understood by the filtering mechanism are listed in the table below.

Warning Filter Actions

Action	Meaning
error	Turn the warning into an exception.
ignore	Discard the warning.
always	Always emit a warning.
default	Print the warning the first time it is generated from each location.
module	Print the warning the first time it is generated from each module.
once	Print the warning the first time it is generated.

Generating Warnings

The simplest way to emit a warning is to call `warn()` with the message as an argument.

```
# warnings_warn.py

import warnings

print('Before the warning')
warnings.warn('This is a warning message')
print('After the warning')
```

Then, when the program runs, the message is printed.

```
$ python3 -u warnings_warn.py

Before the warning
warnings_warn.py:13: UserWarning: This is a warning message
  warnings.warn('This is a warning message')
After the warning
```

Even though the warning is printed, the default behavior is to continue past that point and run the rest of the program. That

Even though the warning is printed, the default behavior is to continue past that point and run the rest of the program. That behavior can be changed with a filter.

```
# warnings_warn_raise.py

import warnings

warnings.simplefilter('error', UserWarning)

print('Before the warning')
warnings.warn('This is a warning message')
print('After the warning')
```

In this example, the `simplefilter()` function adds an entry to the internal filter list to tell the warnings module to raise an exception when a `UserWarning` warning is issued.

```
$ python3 -u warnings_warn_raise.py

Before the warning
Traceback (most recent call last):
  File "warnings_warn_raise.py", line 15, in <module>
    warnings.warn('This is a warning message')
UserWarning: This is a warning message
```

The filter behavior can also be controlled from the command line by using the `-W` option to the interpreter. Specify the filter properties as a string with the five parts (action, message, category, module, and line number) separated by colons (:). For example, if `warnings_warn.py` is run with a filter set to raise an error on `UserWarning`, an exception is produced.

```
$ python3 -u -W "error::UserWarning:::0" warnings_warn.py

Before the warning
Traceback (most recent call last):
  File "warnings_warn.py", line 13, in <module>
    warnings.warn('This is a warning message')
UserWarning: This is a warning message
```

Since the fields for message and module were left blank, they were interpreted as matching anything.

Filtering with Patterns

To filter on more complex rules programmatically, use `filterwarnings()`. For example, to filter based on the content of the message text, give a regular expression pattern as the message argument.

```
# warnings_filterwarnings_message.py

import warnings

warnings.filterwarnings('ignore', '.*do not.*',)

warnings.warn('Show this message')
warnings.warn('Do not show this message')
```

The pattern contains “do not”, but the actual message uses “Do not”. The pattern matches because the regular expression is always compiled to look for case insensitive matches.

```
$ python3 warnings_filterwarnings_message.py

warnings_filterwarnings_message.py:14: UserWarning: Show this
message
  warnings.warn('Show this message')
```

The example program below generates two warnings.

```
# warnings_filter.py

import warnings

warnings.warn('Show this message')
```

```
warnings.warn('Do not show this message')
```

One of the warnings can be ignored using the filter argument on the command line.

```
$ python3 -W "ignore:do not:UserWarning:0" warnings_filter.py

warnings_filter.py:12: UserWarning: Show this message
  warnings.warn('Show this message')
```

The same pattern matching rules apply to the name of the source module containing the call generating the warning. Suppress all messages from the `warnings_filter` module by passing the module name as the pattern to the module argument.

```
# warnings_filterwarnings_module.py

import warnings

warnings.filterwarnings(
    'ignore',
    '.*',
    UserWarning,
    'warnings_filter',
)

import warnings_filter
```

Since the filter is in place, no warnings are emitted when `warnings_filter` is imported.

```
$ python3 warnings_filterwarnings_module.py
```

To suppress only the message on line 13 of `warnings_filter`, include the line number as the last argument to `filterwarnings()`. Use the actual line number from the source file to limit the filter, or 0 to have the filter apply to all occurrences of the message.

```
# warnings_filterwarnings_lineno.py

import warnings

warnings.filterwarnings(
    'ignore',
    '.*',
    UserWarning,
    'warnings_filter',
    13,
)

import warnings_filter
```

The pattern matches any message, so the important arguments are the module name and line number.

```
$ python3 warnings_filterwarnings_lineno.py

../warnings_filter.py:12: UserWarning: Show this message
  warnings.warn('Show this message')
```

Repeated Warnings

By default, most types of warnings are only printed the first time they occur in a given location, with “location” defined by the combination of module and line number where the warning is generated.

```
# warnings_repeated.py

import warnings

def function_with_warning():
    warnings.warn('This is a warning!')
```

```

function_with_warning()
function_with_warning()
function_with_warning()

```

This example calls the same function several times, but produces a single warning.

```

$ python3 warnings_repeated.py

warnings_repeated.py:14: UserWarning: This is a warning!
  warnings.warn('This is a warning!')

```

The "once" action can be used to suppress instances of the same message from different locations.

```

# warnings_once.py

import warnings

warnings.simplefilter('once', UserWarning)

warnings.warn('This is a warning!')
warnings.warn('This is a warning!')
warnings.warn('This is a warning!')

```

The message text for all warnings is saved and only unique messages are printed.

```

$ python3 warnings_once.py

warnings_once.py:14: UserWarning: This is a warning!
  warnings.warn('This is a warning!')

```

Similarly, "module" will suppress repeated messages from the same module, no matter what line number.

Alternate Message Delivery Functions

Normally warnings are printed to `sys.stderr`. Change that behavior by replacing the `showwarning()` function inside the `warnings` module. For example, to send warnings to a log file instead of standard error, replace `showwarning()` with a function that logs the warning.

```

# warnings_showwarning.py

import warnings
import logging

def send_warnings_to_log(message, category, filename, lineno,
                        file=None, line=None):
    logging.warning(
        '%S:%S: %S:%S',
        filename, lineno,
        category.__name__, message,
    )

logging.basicConfig(level=logging.INFO)

old_showwarning = warnings.showwarning
warnings.showwarning = send_warnings_to_log

warnings.warn('message')

```

The warnings are emitted with the rest of the log messages when `warn()` is called.

```

$ python3 warnings_showwarning.py

WARNING:root:warnings_showwarning.py:28: UserWarning:message

```

Formatting

If warnings should go to standard error, but they need to be reformatted, replace `formatwarning()`.

```
# warnings_formatwarning.py

import warnings

def warning_on_one_line(message, category, filename, lineno,
                        file=None, line=None):
    return '-> {}:{}: {}:{}'.format(
        filename, lineno, category.__name__, message)

warnings.warn('Warning message, before')
warnings.formatwarning = warning_on_one_line
warnings.warn('Warning message, after')
```

The format function must return a single string containing the representation of the warning to be displayed to the user.

```
$ python3 -u warnings_formatwarning.py

warnings_formatwarning.py:19: UserWarning: Warning message,
before
  warnings.warn('Warning message, before')
-> warnings_formatwarning.py:21: UserWarning:Warning message,
after
```

Stack Level in Warnings

By default, the warning message includes the source line that generated it, when available. It is not always useful to see the line of code with the actual warning message, though. Instead, `warn()` can be told how far up the stack it has to go to find the line that called the function containing the warning. That way, users of a deprecated function can see where the function is called, instead of the implementation of the function.

```
# warnings_warn_stacklevel.py

1  #!/usr/bin/env python3
2  # encoding: utf-8
3
4  import warnings
5
6
7  def old_function():
8      warnings.warn(
9          'old_function() is deprecated, use new_function()',
10         stacklevel=2)
11
12
13  def caller_of_old_function():
14      old_function()
15
16
17  caller_of_old_function()
```

In this example `warn()` needs to go up the stack two levels, one for itself and one for `old_function()`.

```
$ python3 warnings_warn_stacklevel.py

warnings_warn_stacklevel.py:14: UserWarning: old_function() is deprecated,
use new_function()
  old_function()
```

- [Standard library documentation for warnings](#)
- [PEP 230](#) – Warning Framework
- exceptions – Base classes for exceptions and warnings.
- [logging](#) – An alternative mechanism for delivering warnings is to write to the log.

[Language Tools](#)

[abc — Abstract Base Classes](#)

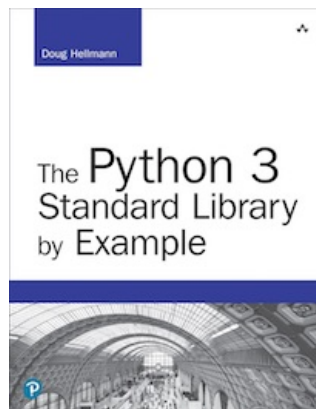
Quick Links

Categories and Filtering
Generating Warnings
Filtering with Patterns
Repeated Warnings
Alternate Message Delivery Functions
Formatting
Stack Level in Warnings

This page was last updated 2018-03-18.

Navigation

[Language Tools](#)
[abc — Abstract Base Classes](#)





[Get the book](#)

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



© Copyright 2019, Doug Hellmann



Other Writing

 Blog
 The Python Standard Library By Example