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

This module provides a standard interface to extract, format and print stack traces of Python programs. It exactly mimics the behavior of the Python interpreter when it prints a stack trace. This is useful when you want to print stack traces under program control, such as in a "wrapper" around the interpreter.

The module uses traceback objects — this is the object type that is stored in the variables `sys.exc_traceback` (deprecated) and `sys.last_traceback` and returned as the third item from `sys.exc_info()`.

The module defines the following functions:

`traceback.`**`print_tb`**(*tb*[, *limit*[, *file*]])

Print up to *limit* stack trace entries from the traceback object *tb*. If *limit* is omitted or `None`, all entries are printed. If *file* is omitted or `None`, the output goes to `sys.stderr`; otherwise it should be an open file or file-like object to receive the output.

`traceback.`**`print_exception`**(*etype*, *value*, *tb*[, *limit*[, *file*]])

Print exception information and up to *limit* stack trace entries from the traceback *tb* to *file*. This differs from `print_tb()` in the following ways: (1) if *tb* is not `None`, it prints a header `Traceback (most recent call last):`; (2) it prints the exception *etype* and *value* after the stack trace; (3) if *etype* is `SyntaxError` and *value* has the appropriate format, it prints the line where the syntax error occurred with a caret indicating the approximate position of the error.

`traceback.`**`print_exc`**([*limit*[, *file*]])

This is a shorthand for `print_exception(sys.exc_type, sys.exc_value, sys.exc_traceback, limit, file)`. (In fact, it uses `sys.exc_info()` to retrieve the same information in a thread-safe way instead of using the deprecated variables.)

`traceback.`**`format_exc`**([*limit*])

This is like `print_exc(limit)` but returns a string instead of printing to a file.

New in version 2.4.

`traceback.`**`print_last`**([*limit*[, *file*]])

This is a shorthand for `print_exception(sys.last_type, sys.last_value, sys.last_traceback, limit, file)`. In general it will work only after an exception has reached an interactive prompt (see `sys.last_type`).

`traceback.`**`print_stack`**([*f*[, *limit*[, *file*]]])

This function prints a stack trace from its invocation point. The optional *f* argument can be used to specify an alternate stack frame to start. The optional limit* and *file* arguments have the same meaning as for `print_exception()`.

traceback.**extract_tb**(*tb*[, *limit*])

Return a list of up to *limit* “pre-processed” stack trace entries extracted from the traceback object *tb*. It is useful for alternate formatting of stack traces. If *limit* is omitted or `None`, all entries are extracted. A “pre-processed” stack trace entry is a 4-tuple (*filename*, *line number*, function name*, *text*) representing the information that is usually printed for a stack trace. The *text* is a string with leading and trailing whitespace stripped; if the source is not available it is `None`.

traceback.**extract_stack**([*f*[, *limit*]])

Extract the raw traceback from the current stack frame. The return value has the same format as for `extract_tb()`. The optional *f* and *limit* arguments have the same meaning as for `print_stack()`.

traceback.**format_list**(*extracted_list*)

Given a list of tuples as returned by `extract_tb()` or `extract_stack()`, return a list of strings ready for printing. Each string in the resulting list corresponds to the item with the same index in the argument list. Each string ends in a newline; the strings may contain internal newlines as well, for those items whose source text line is not `None`.

traceback.**format_exception_only**(*etype*, *value*)

Format the exception part of a traceback. The arguments are the exception type, *etype* and *value* such as given by `sys.last_type` and `sys.last_value`. The return value is a list of strings, each ending in a newline. Normally, the list contains a single string; however, for `SyntaxError` exceptions, it contains several lines that (when printed) display detailed information about where the syntax error occurred. The message indicating which exception occurred is the always last string in the list.

traceback.**format_exception**(*etype*, *value*, *tb*[, *limit*])

Format a stack trace and the exception information. The arguments have the same meaning as the corresponding arguments to `print_exception()`. The return value is a list of strings, each ending in a newline and some containing internal newlines. When these lines are concatenated and printed, exactly the same text is printed as does `print_exception()`.

traceback.**format_tb**(*tb*[, *limit*])

A shorthand for `format_list(extract_tb(tb, limit))`.

traceback.**format_stack**([*f*[, *limit*]])

A shorthand for `format_list(extract_stack(f, limit))`.

traceback.**tb_lineno**(*tb*)

This function returns the current line number set in the traceback object. This function was necessary because in versions of Python prior to 2.3 when the `-O` flag was passed to Python the `tb.tb_lineno` was not updated correctly. This function has no use in versions past 2.3.

28.10.1. Traceback Examples

This simple example implements a basic read-eval-print loop, similar to (but less useful than) the standard Python interactive interpreter loop. For a more complete implementation of the interpreter

loop, refer to the `code` module.

```
import sys, traceback

def run_user_code(envdir):
    source = raw_input(">>> ")
    try:
        exec source in envdir
    except:
        print "Exception in user code:"
        print '-'*60
        traceback.print_exc(file=sys.stdout)
        print '-'*60

envdir = {}
while 1:
    run_user_code(envdir)
```

The following example demonstrates the different ways to print and format the exception and traceback:

```
import sys, traceback

def lumberjack():
    bright_side_of_death()

def bright_side_of_death():
    return tuple()[0]

try:
    lumberjack()
except IndexError:
    exc_type, exc_value, exc_traceback = sys.exc_info()
    print "*** print_tb:"
    traceback.print_tb(exc_traceback, limit=1, file=sys.stdout)
    print "*** print_exception:"
    traceback.print_exception(exc_type, exc_value, exc_traceback,
                              limit=2, file=sys.stdout)
    print "*** print_exc:"
    traceback.print_exc()
    print "*** format_exc, first and last line:"
    formatted_lines = traceback.format_exc().splitlines()
    print formatted_lines[0]
    print formatted_lines[-1]
    print "*** format_exception:"
    print repr(traceback.format_exception(exc_type, exc_value,
                                          exc_traceback))
    print "*** extract_tb:"
    print repr(traceback.extract_tb(exc_traceback))
    print "*** format_tb:"
    print repr(traceback.format_tb(exc_traceback))
    print "*** tb_lineno:", exc_traceback.tb_lineno
```

The output for the example would look similar to this:

```
*** print_tb:
  File "<doctest...>", line 10, in <module>
    lumberjack()
*** print_exception:
Traceback (most recent call last):
  File "<doctest...>", line 10, in <module>
```

```
    lumberjack()
  File "<doctest...>", line 4, in lumberjack
    bright_side_of_death()
IndexError: tuple index out of range
*** print_exc:
Traceback (most recent call last):
  File "<doctest...>", line 10, in <module>
    lumberjack()
  File "<doctest...>", line 4, in lumberjack
    bright_side_of_death()
IndexError: tuple index out of range
*** format_exc, first and last line:
Traceback (most recent call last):
IndexError: tuple index out of range
*** format_exception:
['Traceback (most recent call last):\n',
 '  File "<doctest...>", line 10, in <module>\n    lumberjack()\n',
 '  File "<doctest...>", line 4, in lumberjack\n    bright_side_of_death()\n',
 '  File "<doctest...>", line 7, in bright_side_of_death\n    return tuple()[0]\n',
 'IndexError: tuple index out of range\n']
*** extract_tb:
[('<doctest...>', 10, '<module>', 'lumberjack()'),
 ('<doctest...>', 4, 'lumberjack', 'bright_side_of_death()'),
 ('<doctest...>', 7, 'bright_side_of_death', 'return tuple()[0]')]
*** format_tb:
['  File "<doctest...>", line 10, in <module>\n    lumberjack()\n',
 '  File "<doctest...>", line 4, in lumberjack\n    bright_side_of_death()\n',
 '  File "<doctest...>", line 7, in bright_side_of_death\n    return tuple()[0]\n']
*** tb_lineno: 10
```

The following example shows the different ways to print and format the stack:

```
>>> import traceback
>>> def another_function():
...     lumberstack()
...
>>> def lumberstack():
...     traceback.print_stack()
...     print repr(traceback.extract_stack())
...     print repr(traceback.format_stack())
...
>>> another_function()
  File "<doctest>", line 10, in <module>
    another_function()
  File "<doctest>", line 3, in another_function
    lumberstack()
  File "<doctest>", line 6, in lumberstack
    traceback.print_stack()
[('<doctest>', 10, '<module>', 'another_function()'),
 ('<doctest>', 3, 'another_function', 'lumberstack()'),
 ('<doctest>', 7, 'lumberstack', 'print repr(traceback.extract_stack())')]
['  File "<doctest>", line 10, in <module>\n    another_function()\n',
 '  File "<doctest>", line 3, in another_function\n    lumberstack()\n',
 '  File "<doctest>", line 8, in lumberstack\n    print repr(traceback.format_stack())\n']
```

This last example demonstrates the final few formatting functions:

```
>>> import traceback
>>> traceback.format_list([('spam.py', 3, '<module>', 'spam.eggs()'),
...                        ('eggs.py', 42, 'eggs', 'return "bacon"')])
['  File "spam.py", line 3, in <module>\n    spam.eggs()\n',
 '  File "eggs.py", line 42, in eggs\n    return "bacon"\n']
```

```
>>> an_error = IndexError('tuple index out of range')
>>> traceback.format_exception_only(type(an_error), an_error)
['IndexError: tuple index out of range\n']
```