cgitb — Detailed Traceback Reports

Purpose: cgitb provides more detailed traceback information than traceback.

cgitb is a valuable debugging tool in the standard library. It was originally designed for showing errors and debugging information in web applications and was later updated to include plain-text output as well, but unfortunately was never renamed. This has led to obscurity, and the module is not used as often as it could be.

Standard Traceback Dumps

Python’s default exception handling behavior is to print a traceback to the standard error output stream with the call stack leading up to the error position. This basic output frequently contains enough information to understand the cause of the exception and permit a fix.

```
# cgitb_basic_traceback.py

def func2(a, divisor):
    return a / divisor


def func1(a, b):
    c = b - 5
    return func2(a, c)

func1(1, 5)
```

This sample program has a subtle error in `func2()`.

```
$ python3 cgitb_basic_traceback.py

Traceback (most recent call last):
  File "cgitb_basic_traceback.py", line 18, in <module>
    func1(1, 5)
  File "cgitb_basic_traceback.py", line 16, in func1
    return func2(a, c)
  File "cgitb_basic_traceback.py", line 11, in func2
    return a / divisor
ZeroDivisionError: division by zero
```

Enabling Detailed Tracebacks

While the basic traceback includes enough information to spot the error, enabling `cgitb` gives more detail. `cgitb` replaces `sys.excepthook` with a function that gives extended tracebacks.

```
# cgitb_local_vars.py

import cgitb
cgitb.enable(format='text')
```

The error report from this example is much more extensive than the original. Each frame of the stack is listed, along with:

- The full path to the source file, instead of just the base name
- The values of the arguments to each function in the stack
- A few lines of source context from around the line in the error path
- The values of variables in the expression causing the error

Having access to the variables involved in the error stack can help find a logical error that occurs somewhere higher in the stack than the line where the actual exception is generated.

```
$ python3 cgitb_local_vars.py

ZeroDivisionError
```

```
ZeroDivisionError
Python 3.7.1: .../bin/python3
Sun Dec  9 10:46:17 2018

A problem occurred in a Python script.  Here is the sequence of
function calls leading up to the error, in the order they
occurred.

 .../cgitb_local_vars.py in <module>()
   18 def func1(a, b):
   19     c = b - 5
   20     return func2(a, c)
   21
   22 func1(1, 5)
func1 = <function func1>

 .../cgitb_local_vars.py in func1(a=1, b=5)
   18 def func1(a, b):
   19     c = b - 5
   20     return func2(a, c)
   21
   22 func1(1, 5)
global func2 = <function func2>
a = 1
c = 0

 .../cgitb_local_vars.py in func2(a=1, divisor=0)
   13
   14 def func2(a, divisor):
   15     return a / divisor
   16
   17
a = 1
divisor = 0
ZeroDivisionError: division by zero
    __cause__ = None
    __class__ = <class 'ZeroDivisionError'>
    __context__ = None
    __delattr__ = <method-wrapper '__delattr__' of
    ZeroDivisionError object>
    __dict__ = {}
    __dir__ = <built-in method __dir__ of ZeroDivisionError
    object>
    __doc__ = 'Second argument to a division or modulo operation
    was zero.'
    __eq__ = <method-wrapper '__eq__' of ZeroDivisionError
    object>
    __format__ = <built-in method __format__ of
    ZeroDivisionError object>
    __ge__ = <method-wrapper '__ge__' of ZeroDivisionError
    object>
    __getattribute__ = <method-wrapper '__getattribute__' of
    ZeroDivisionError object>
    __gt__ = <method-wrapper '__gt__' of ZeroDivisionError
    object>
    __hash__ = <method-wrapper '__hash__' of ZeroDivisionError
    object>
    __init__ = <method-wrapper '__init__' of ZeroDivisionError
    object>
    __init_subclass__ = <built-in method __init_subclass__ of
    type object>
    __le__ = <method-wrapper '__le__' of ZeroDivisionError
    object>
    __lt__ = <method-wrapper '__lt__' of ZeroDivisionError
    object>
    __ne__ = <method-wrapper '__ne__' of ZeroDivisionError
    object>
    __new__ = <built-in method __new__ of type object>
    __reduce__ = <built-in method __reduce__ of
    ZeroDivisionError object>
    __reduce_ex__ = <built-in method __reduce_ex__ of
    ZeroDivisionError object>
    __repr__ = <method-wrapper '__repr__' of ZeroDivisionError
    object>
    __setattr__ = <method-wrapper '__setattr__' of
    ZeroDivisionError object>
    __setstate__ = <built-in method __setstate__ of
    ZeroDivisionError object>
    __sizeof__ = <built-in method __sizeof__ of
    ZeroDivisionError object>
    __str__ = <method-wrapper '__str__' of ZeroDivisionError
    object>
    __subclasshook__ = <built-in method __subclasshook__ of type
    object>
    __suppress_context__ = False
    __traceback__ = <traceback object>
    args = ('division by zero',)
    with_traceback = <built-in method with_traceback of
    ZeroDivisionError object>

The above is a description of an error in a Python program.
Here is
the original traceback:

Traceback (most recent call last):
  File "cgitb_local_vars.py", line 22, in <module>
    func1(1, 5)
  File "cgitb_local_vars.py", line 20, in func1
    return func2(a, c)
  File "cgitb_local_vars.py", line 15, in func2
    return a / divisor
ZeroDivisionError: division by zero
```

In the case of this code with a `ZeroDivisionError`, it is apparent that the problem is introduced in the computation of the value of c in `func1()`, rather than where the value is used in `func2()`.

The end of the output also includes the full details of the exception object (in case it has attributes other than `message` that would be useful for debugging) and the original form of a traceback dump.

Local Variables in Tracebacks

The code in cgitb that examines the variables used in the stack frame leading to the error is smart enough to evaluate object attributes to display them, too.

```python
# cgitb_with_classes.py

import cgitb
cgitb.enable(format='text', context=12)


class BrokenClass:
    """This class has an error.
    """

    def __init__(self, a, b):
        """Be careful passing arguments in here.
        """
        self.a = a
        self.b = b
        self.c = self.a * self.b
        # Really
        # long
        # comment
        # goes
        # here.
        self.d = self.a / self.b
        return

o = BrokenClass(1, 0)
```

If a function or method includes a lot of in-line comments, whitespace, or other code that makes it very long, then having the default of five lines of context may not provide enough direction. When the body of the function is pushed out of the code window displayed, there is not enough context to understand the location of the error. Using a larger context value with `cgitb` solves this problem. Passing an integer as the `context` argument to `enable()` controls the amount of code displayed for each line of the traceback.

This output shows that `self.a` and `self.b` are involved in the error-prone code.

```
$ python3 cgitb_with_classes.py

ZeroDivisionError
Python 3.7.1: .../bin/python3
Sun Dec  9 10:46:17 2018

A problem occurred in a Python script.  Here is the sequence of
function calls leading up to the error, in the order they
occurred.

 .../cgitb_with_classes.py in <module>()
   21           self.a = a
   22           self.b = b
   23           self.c = self.a * self.b
   24           # Really
   25           # long
   26           # comment
   27           # goes
   28           # here.
   29           self.d = self.a / self.b
   30           return
   31
   32 o = BrokenClass(1, 0)
o undefined
BrokenClass = <class '__main__.BrokenClass'>

 .../cgitb_with_classes.py in
 __init__(self=<__main__.BrokenClass object>, a=1, b=0)
   21           self.a = a
   22           self.b = b
   23           self.c = self.a * self.b
   24           # Really
   25           # long
   26           # comment
   27           # goes
   28           # here.
   29           self.d = self.a / self.b
   30           return
   31
   32 o = BrokenClass(1, 0)
self = <__main__.BrokenClass object>
self.d undefined
self.a = 1
self.b = 0
ZeroDivisionError: division by zero
    __cause__ = None
    __class__ = <class 'ZeroDivisionError'>
    __context__ = None
    __delattr__ = <method-wrapper '__delattr__' of
    ZeroDivisionError object>
    __dict__ = {}
    __dir__ = <built-in method __dir__ of ZeroDivisionError
    object>
    __doc__ = 'Second argument to a division or modulo operation
    was zero.'
    __eq__ = <method-wrapper '__eq__' of ZeroDivisionError
    object>
    __format__ = <built-in method __format__ of
    ZeroDivisionError object>
    __ge__ = <method-wrapper '__ge__' of ZeroDivisionError
    object>
    __getattribute__ = <method-wrapper '__getattribute__' of
    ZeroDivisionError object>
    __gt__ = <method-wrapper '__gt__' of ZeroDivisionError
    object>
    __hash__ = <method-wrapper '__hash__' of ZeroDivisionError
    object>
    __init__ = <method-wrapper '__init__' of ZeroDivisionError
    object>
    __init_subclass__ = <built-in method __init_subclass__ of
    type object>
    __le__ = <method-wrapper '__le__' of ZeroDivisionError
    object>
    __lt__ = <method-wrapper '__lt__' of ZeroDivisionError
    object>
    __ne__ = <method-wrapper '__ne__' of ZeroDivisionError
    object>
    __new__ = <built-in method __new__ of type object>
    __reduce__ = <built-in method __reduce__ of
    ZeroDivisionError object>
    __reduce_ex__ = <built-in method __reduce_ex__ of
    ZeroDivisionError object>
    __repr__ = <method-wrapper '__repr__' of ZeroDivisionError
    object>
    __setattr__ = <method-wrapper '__setattr__' of
    ZeroDivisionError object>
    __setstate__ = <built-in method __setstate__ of
    ZeroDivisionError object>
    __sizeof__ = <built-in method __sizeof__ of
    ZeroDivisionError object>
    __str__ = <method-wrapper '__str__' of ZeroDivisionError
    object>
    __subclasshook__ = <built-in method __subclasshook__ of type
    object>
    __suppress_context__ = False
    __traceback__ = <traceback object>
    args = ('division by zero',)
    with_traceback = <built-in method with_traceback of
    ZeroDivisionError object>

The above is a description of an error in a Python program.
Here is
the original traceback:

Traceback (most recent call last):
  File "cgitb_with_classes.py", line 32, in <module>
    o = BrokenClass(1, 0)
  File "cgitb_with_classes.py", line 29, in __init__
    self.d = self.a / self.b
ZeroDivisionError: division by zero
```

Exception Properties

In addition to the local variables from each stack frame, cgitb shows all properties of the exception object. Extra properties on custom exception types are printed as part of the error report.

```python
# cgitb_exception_properties.py

import cgitb
cgitb.enable(format='text')


class MyException(Exception):
    """Add extra properties to a special exception
    """

    def __init__(self, message, bad_value):
        self.bad_value = bad_value
        Exception.__init__(self, message)
        return

raise MyException('Normal message', bad_value=99)
```

In this example, the bad_value property is included along with the standard message and args values.

```
$ python3 cgitb_exception_properties.py

MyException
Python 3.7.1: .../bin/python3
Sun Dec  9 10:46:17 2018

A problem occurred in a Python script.  Here is the sequence of
function calls leading up to the error, in the order they
occurred.

 .../cgitb_exception_properties.py in <module>()
   19         self.bad_value = bad_value
   20         Exception.__init__(self, message)
   21         return
   22
   23 raise MyException('Normal message', bad_value=99)
MyException = <class '__main__.MyException'>
bad_value undefined
MyException: Normal message
    __cause__ = None
    __class__ = <class '__main__.MyException'>
    __context__ = None
    __delattr__ = <method-wrapper '__delattr__' of MyException
    object>
    __dict__ = {'bad_value': 99}
    __dir__ = <built-in method __dir__ of MyException object>
    __doc__ = 'Add extra properties to a special exception\n
    '
    __eq__ = <method-wrapper '__eq__' of MyException object>
    __format__ = <built-in method __format__ of MyException
    object>
    __ge__ = <method-wrapper '__ge__' of MyException object>
    __getattribute__ = <method-wrapper '__getattribute__' of
    MyException object>
    __gt__ = <method-wrapper '__gt__' of MyException object>
    __hash__ = <method-wrapper '__hash__' of MyException object>
    __init__ = <bound method MyException.__init__ of
    MyException('Normal message')>
    __init_subclass__ = <built-in method __init_subclass__ of
    type object>
    __le__ = <method-wrapper '__le__' of MyException object>
    __lt__ = <method-wrapper '__lt__' of MyException object>
    __module__ = '__main__'
    __ne__ = <method-wrapper '__ne__' of MyException object>
    __new__ = <built-in method __new__ of type object>
    __reduce__ = <built-in method __reduce__ of MyException
    object>
    __reduce_ex__ = <built-in method __reduce_ex__ of
    MyException object>
    __repr__ = <method-wrapper '__repr__' of MyException object>
    __setattr__ = <method-wrapper '__setattr__' of MyException
    object>
    __setstate__ = <built-in method __setstate__ of MyException
    object>
    __sizeof__ = <built-in method __sizeof__ of MyException
    object>
    __str__ = <method-wrapper '__str__' of MyException object>
    __subclasshook__ = <built-in method __subclasshook__ of type
    object>
    __suppress_context__ = False
    __traceback__ = <traceback object>
    __weakref__ = None
    args = ('Normal message',)
    bad_value = 99
    with_traceback = <built-in method with_traceback of
    MyException object>

The above is a description of an error in a Python program.
Here is
the original traceback:

Traceback (most recent call last):
  File "cgitb_exception_properties.py", line 23, in <module>
    raise MyException('Normal message', bad_value=99)
MyException: Normal message
```

HTML Output

Because cgitb was originally developed for handling exceptions in web applications, no discussion would be complete without mentioning its original HTML output format. The earlier examples all shows plain text output. To produce HTML instead, leave out the `format` argument (or specify `"html"`). Most modern web applications are constructed using a framework that includes an error reporting facility, so the HTML form is largely obsolete.

Logging Tracebacks

For many situations, printing the traceback details to standard error is the best resolution. In a production system, however, logging the errors is even better. The `enable()` function includes an optional argument, `logdir`, to enable error logging. When a directory name is provided, each exception is logged to its own file in the given directory.

```
# cgitb_log_exception.py

import cgitb
import os

LOGDIR = os.path.join(os.path.dirname(__file__), 'LOGS')

if not os.path.exists(LOGDIR):
    os.makedirs(LOGDIR)

cgitb.enable(
    logdir=LOGDIR,
    display=False,
    format='text',
)


def func(a, divisor):
    return a / divisor

func(1, 0)
```

Even though the error display is suppressed, a message is printed describing where to go to find the error log.

```
$ python3 cgitb_log_exception.py

<p>A problem occurred in a Python script.
.../LOGS/tmpdl2oafqt.txt contains the description of this error.

$ ls LOGS

tmpdl2oafqt.txt

$ cat LOGS/*.txt

ZeroDivisionError
Python 3.7.1: .../bin/python3
Sun Dec  9 10:46:17 2018

A problem occurred in a Python script.  Here is the sequence of
function calls leading up to the error, in the order they
occurred.

 .../cgitb_log_exception.py in <module>()
   24
   25 def func(a, divisor):
   26     return a / divisor
   27
   28 func(1, 0)
func = <function func>

    .../cgitb_log_exception.py in func(a=1, divisor=0)
 .../cgitb_log_exception.py in func(a=1, divisor=0)
   24
   25 def func(a, divisor):
   26     return a / divisor
   27
   28 func(1, 0)
a = 1
divisor = 0
ZeroDivisionError: division by zero
    __cause__ = None
    __class__ = <class 'ZeroDivisionError'>
    __context__ = None
    __delattr__ = <method-wrapper '__delattr__' of
    ZeroDivisionError object>
    __dict__ = {}
    __dir__ = <built-in method __dir__ of ZeroDivisionError
    object>
    __doc__ = 'Second argument to a division or modulo operation
    was zero.'
    __eq__ = <method-wrapper '__eq__' of ZeroDivisionError
    object>
    __format__ = <built-in method __format__ of
    ZeroDivisionError object>
    __ge__ = <method-wrapper '__ge__' of ZeroDivisionError
    object>
    __getattribute__ = <method-wrapper '__getattribute__' of
    ZeroDivisionError object>
    __gt__ = <method-wrapper '__gt__' of ZeroDivisionError
    object>
    __hash__ = <method-wrapper '__hash__' of ZeroDivisionError
    object>
    __init__ = <method-wrapper '__init__' of ZeroDivisionError
    object>
    __init_subclass__ = <built-in method __init_subclass__ of
    type object>
    __le__ = <method-wrapper '__le__' of ZeroDivisionError
    object>
    __lt__ = <method-wrapper '__lt__' of ZeroDivisionError
    object>
    __ne__ = <method-wrapper '__ne__' of ZeroDivisionError
    object>
    __new__ = <built-in method __new__ of type object>
    __reduce__ = <built-in method __reduce__ of
    ZeroDivisionError object>
    __reduce_ex__ = <built-in method __reduce_ex__ of
    ZeroDivisionError object>
    __repr__ = <method-wrapper '__repr__' of ZeroDivisionError
    object>
    __setattr__ = <method-wrapper '__setattr__' of
    ZeroDivisionError object>
    __setstate__ = <built-in method __setstate__ of
    ZeroDivisionError object>
    __sizeof__ = <built-in method __sizeof__ of
    ZeroDivisionError object>
    __str__ = <method-wrapper '__str__' of ZeroDivisionError
    object>
    __subclasshook__ = <built-in method __subclasshook__ of type
    object>
    __suppress_context__ = False
    __traceback__ = <traceback object>
    args = ('division by zero',)
    with_traceback = <built-in method with_traceback of
    ZeroDivisionError object>

The above is a description of an error in a Python program.
Here is
the original traceback:

Traceback (most recent call last):
  File "cgitb_log_exception.py", line 28, in <module>
    func(1, 0)
  File "cgitb_log_exception.py", line 26, in func
```

```
    return a / divisor
ZeroDivisionError: division by zero
```

See also

- Standard library documentation for cgitb
- `traceback` – Standard library module for working with tracebacks.
- `inspect` – The `inspect` module includes more functions for examining the stack.
- `sys` – The `sys` module provides access to the current exception value and the `excepthook` handler invoked when an exception occurs.
- Improved traceback module – Discussion on the Python development mailing list about improvements to the traceback module and related enhancements other developers use locally.

traceback — Exceptions and Stack Traces pdb — Interactive Debugger

This page was last updated 2018-12-09.

Navigation
traceback — Exceptions and Stack Traces
pdb — Interactive Debugger

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?