

#### The basics

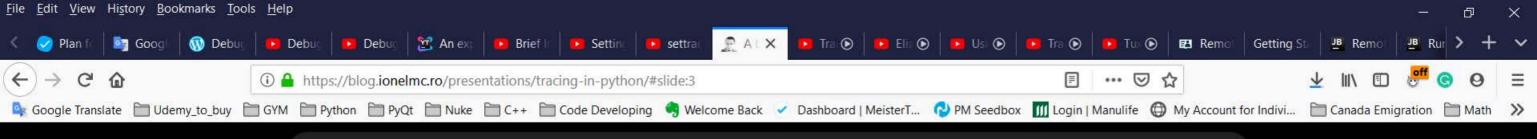
 Python can call a function for every line executed def mytracer(frame, event, args):

```
import sys
sys.settrace(mytracer)
```

• There are 4 kinds of events: call, line, return, exception

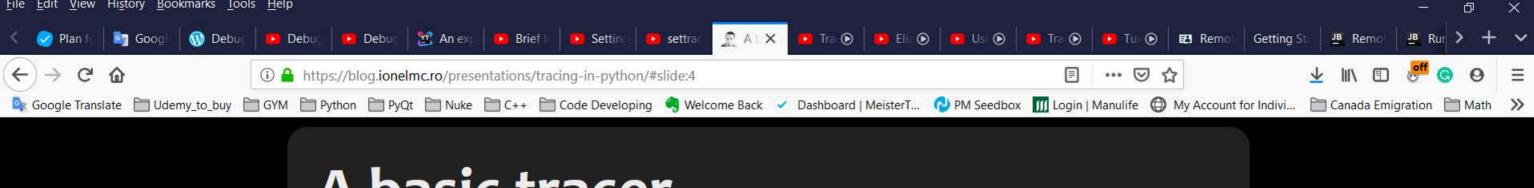
(don't believe the bs in docs about c\_call, c\_return and c\_exception - those are only used for profiling via the similar sys.setprofile)

- The frame object contains information about the code being run (module, filename location and call stack)
- The arg will have the return or exception value (if event is return or exception)



#### A basic tracer

An almost useful tracer:



#### A basic tracer

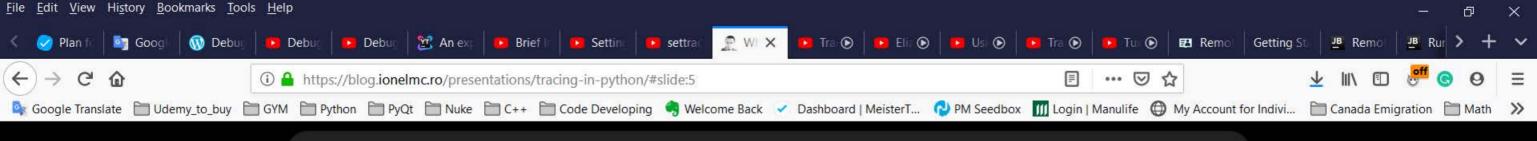
#### Gets you this:

File Edit View History Bookmarks Tools Help

```
call def foobar(a, b):
examples.py:3
examples.py:4
                     line
                               c = a + b
examples.py:5
                     line
                               try:
                     line
examples.py:6
                                   raise Exception(c)
examples.py:6
                exception
                                   raise Exception(c)
examples.py:7
                     line
                               except Exception as exc:
examples.py:8
                     line
                                   raise RuntimeError(exc)
                                   raise RuntimeError(exc)
examples.py:8
                exception
examples.py:10
                     line
                                   c = None
examples.py:10
                   return
                                   c = None
```

#### For some code:

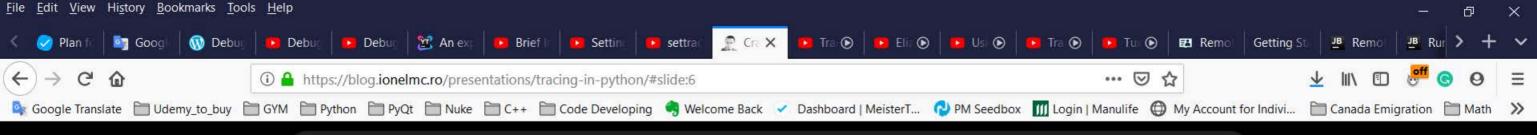
```
def foobar(a, b):
    c = a + b
    try:
        raise Exception(c)
    except Exception as exc:
        raise RuntimeError(exc)
    finally:
        c = None
```



# What other stuff can you do?

Besides printing code, you could implement:

- debuggers (like pdb)
- call graphs
- line profilers
- special purpose logging
- testing tools (coverage)
- crazy stuff



## **Crazy stuff?**

Did you know decorators were added in Python 2.4?

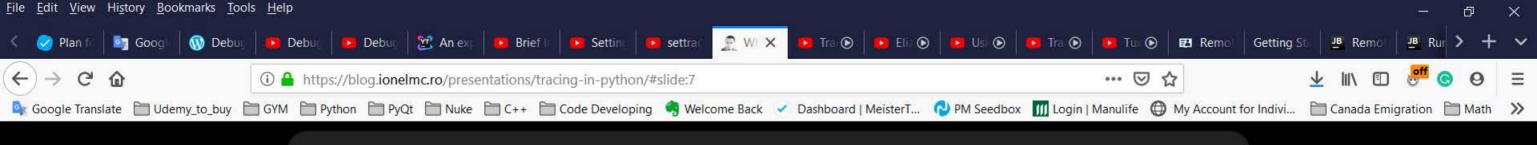
On 2.3 DecoratorTools supports "decorator syntax" with a tracer:

```
from peak.util.decorators import decorate

class Foo(object):
    decorate(classmethod)
    def something(cls,etc):
        """This is a classmethod"""

How?
```

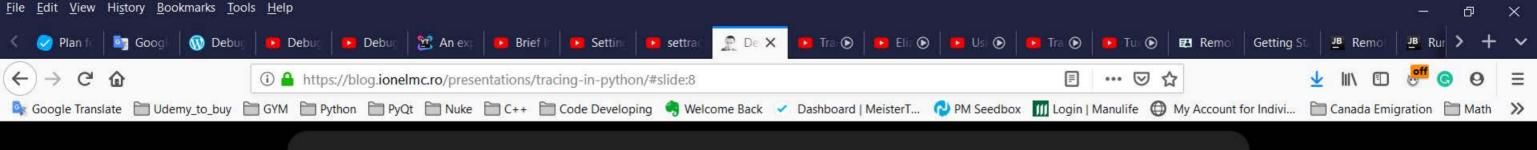
Traces the local scope, diffs locals and patches when the function gets added into them.



# What about profilers

A profiler pretty much the same, only that:

- it isn't called for line events
- return value doesn't matter (it's always "stepped in")



# Dealing with the slowness

There is support to implement a tracer in C:

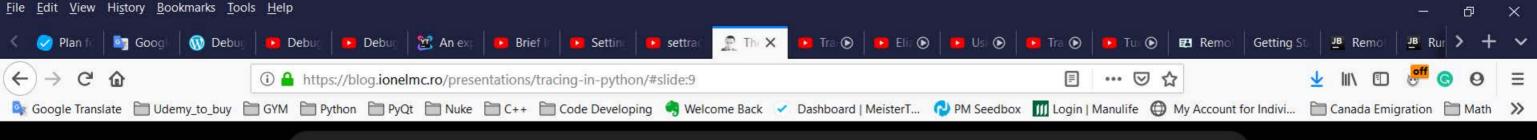
```
PyEval_SetTrace(my_fast_tracer, optional_context_object)
```

What sys.settrace actually do:

```
PyEval_SetTrace(trace_trampoline, args); // args is the function
```

RoPython Cluj — 7 Dec 2017

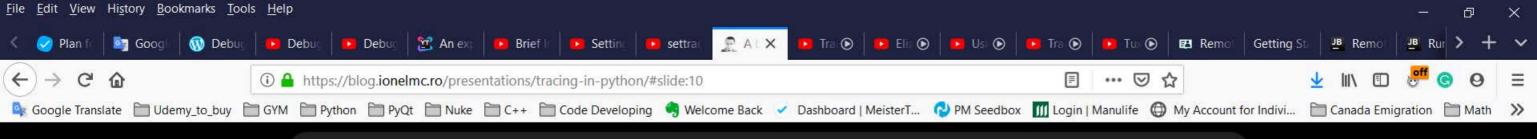
The trampoline starts like this



## The frame object

The most interesting stuff:

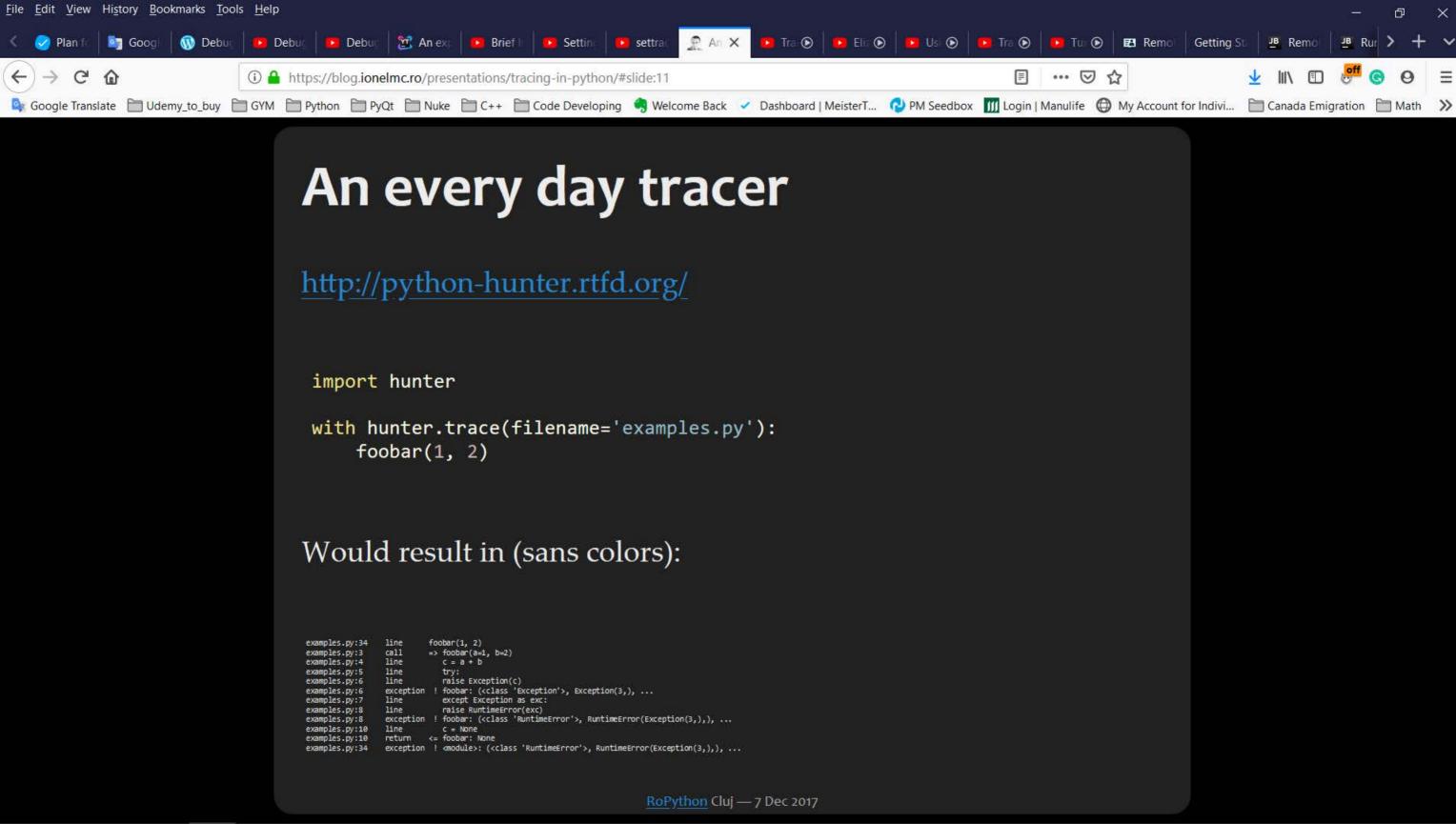
- frame.f\_locals
- frame.f\_globals
- frame.f\_code.co\_name function name
- frame.f\_lineno
- frame.f\_globals['\_\_name\_\_'] module name (could be missing tho)



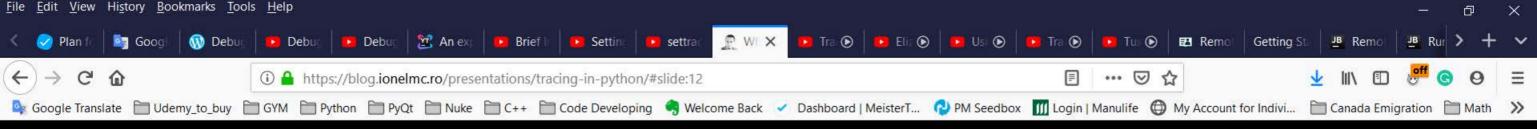
## A better way?

Contrived when adding filters:

```
def dumbtrace(frame, event, args):
    filename = frame.f code.co filename # missing handling for .pyc/o etc
    if filename == 'examples.py':
        sys.stdout.write("%015s:%-3s %09s %s" % (
            filename,
            frame.f lineno,
            event,
            linecache.getline(frame.f code.co filename, frame.f lineno)
        ))
        return dumbtrace # "step in"
sys.settrace(dumbtrace)
try:
    foobar(1, 2)
finally: # why uninstall?
    sys.settrace(None)
```

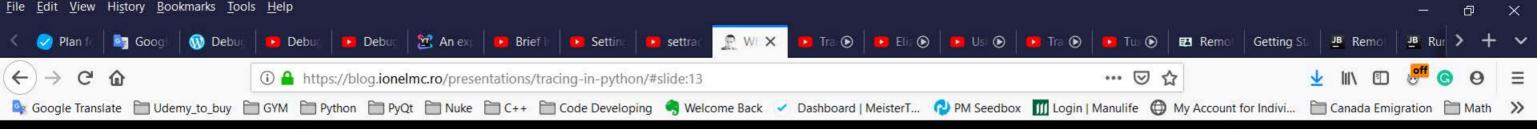


口



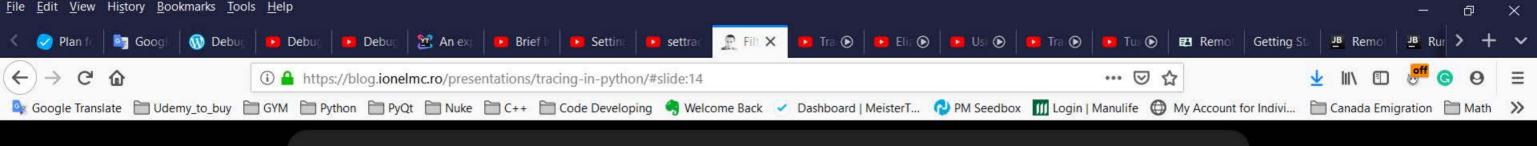
## What can you filter on

- arg A value that depends on kind
- calls A counter for total number of calls up to this Event
- depth Tracing depth (increases on calls, decreases on returns)
- filename A string with absolute path to file.
- fullsource A string with a line or more (decorators are included if it's a class/function definition).
- function A string with function name.
- globals A dict with global variables.
- kind The kind of the event ('call', 'line', 'return' or 'exception').



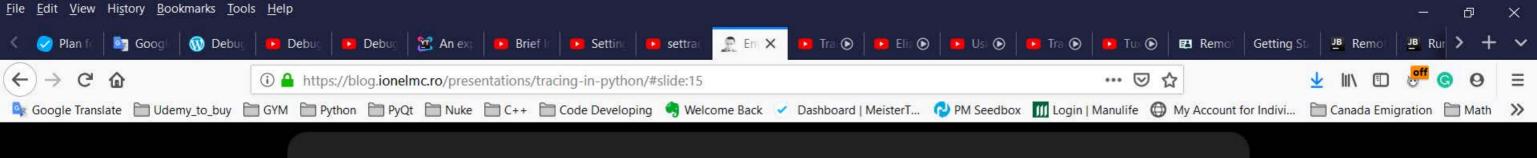
# What can you filter on

- lineno An integer with line number in file.
- locals A dict with local variables.
- module A string with module name (eg "foo.bar").
- source A string with the sourcecode (fast but may be incomplete).
- stdlib A boolean flag. True if frame is in stdlib.
- threadid Current thread ident. If current thread is main thread then it returns None.
- threadname Current thread name.



# Filtering operators

```
hunter.trace(filename='examples.py')
hunter.trace(filename_endswith='examples.py')
hunter.trace(filename_ew='examples.py')
hunter.trace(filename regex=r'[\/]examples\.py')
hunter.trace(filename rx=r'[\/]examples\.py')
hunter.trace(filename contains='examples')
hunter.trace(filename has='examples')
hunter.trace(filename in=['examples.py'])
hunter.trace(depth_lt=10)
hunter.trace(depth_lte=10)
hunter.trace(depth gt=1)
hunter.trace(depth gte=1)
                               RoPython Cluj — 7 Dec 2017
```



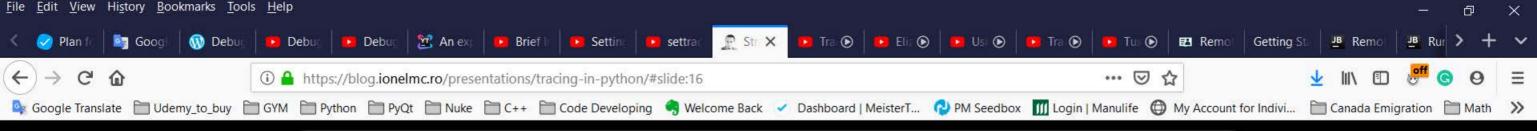
#### **Environment activation**

Bash:

PYTHONHUNTER='module="os.path" | python examples.py

Batch:

set PYTHONHUNTER=module='os.path'
python examples.py



### Strace-like functionality

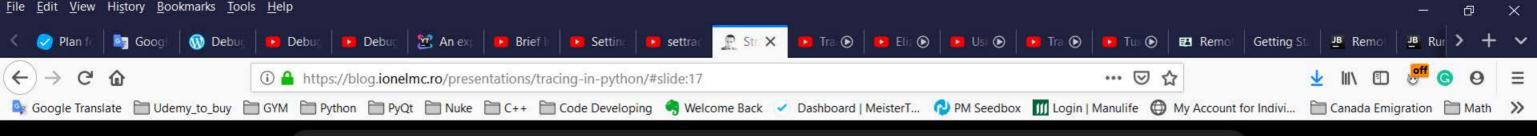
It has it but it's Linux only (probably works on BSD/OSX):

#### Arguments:

- --gdb Use GDB to activate tracing. WARNING: it may deadlock the process!
- -s SIGNAL, --signal SIGNAL

  Send the given SIGNAL to the process before connecting.

The OPTIONS are hunter.trace() arguments.



### Strace-like functionality

How it works:

• GDB: automates it and drops this bomb in the process:

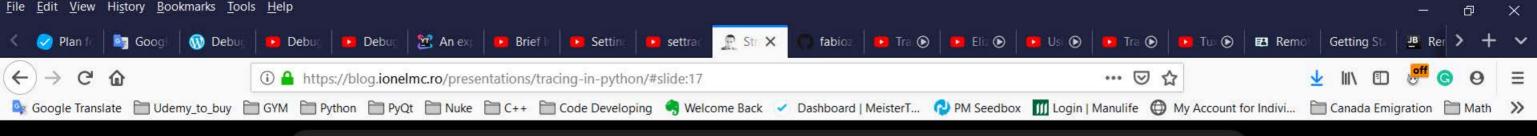
```
Py_AddPendingCall(
         PyRun_SimpleString,
         "from hunter import remote; remote.activate(...)")
```

• Manhole: requires user to install it:

```
from hunter import remote
remote.install()
```

An idea for Windows support:

One way would be using <a href="https://github.com/fabioz/">https://github.com/fabioz/</a>
 /PyDev.Debugger/ for GDB-like attach-inject.



### Strace-like functionality

How it works:

• GDB: automates it and drops this bomb in the process:

```
Py_AddPendingCall(
         PyRun_SimpleString,
         "from hunter import remote; remote.activate(...)")
```

• Manhole: requires user to install it:

```
from hunter import remote
remote.install()
```

An idea for Windows support:

One way would be using <a href="https://github.com/fabioz/">https://github.com/fabioz/</a>
 /PyDev.Debugger/ for GDB-like attach-inject.