profilehooks

The last 3rd party package that we will look at in this chapter is called **profilehooks**. It is a collection of decorators specifically designed for profiling functions. To install profilehooks, just do the following:

```
pip install profilehooks
```

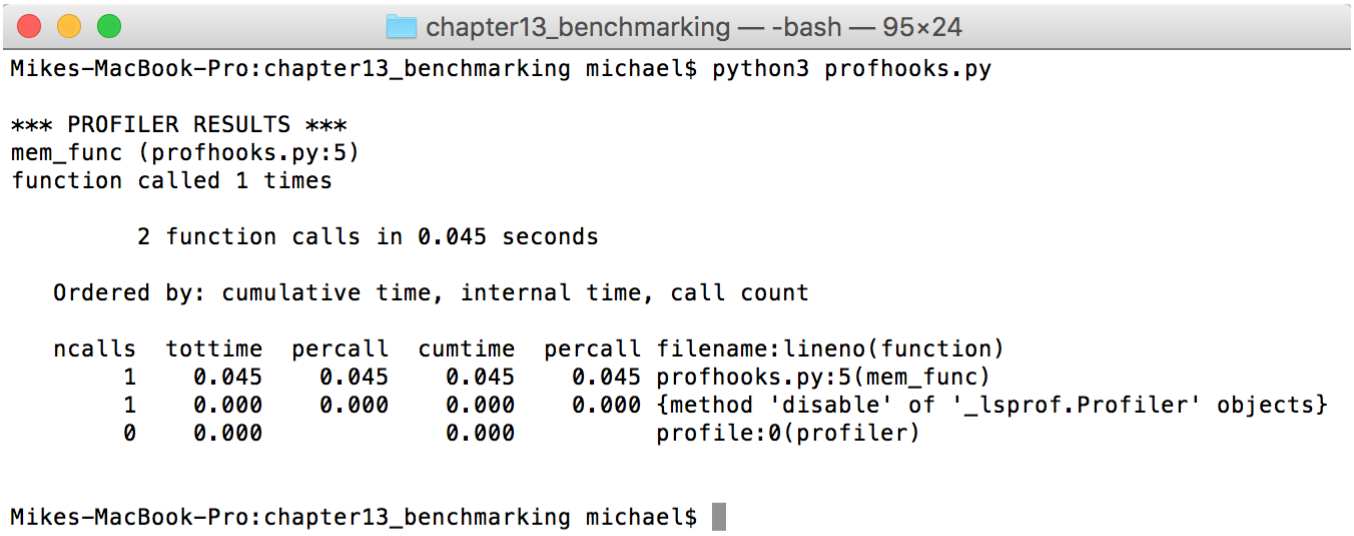
Now that we have it installed, let's re-use the example from the last section and modify it slightly to use profilehooks:

```
# profhooks.py
from profilehooks import profile


#@profile
def mem_func():
    lots_of_numbers = list(range(1500))
    x = ['letters'] * (5 ** 10)
    del lots_of_numbers
    return None

if __name__ == '__main__':
    mem_func()
```

All you need to do to use profilehooks is import it and then decorate the function that you want to profile. If you run the code above, you will get output similar to the following sent to stdout:



The output for this package appears to follow that of the cProfile module from Python's standard library. You can refer to the descriptions of the columns earlier in this chapter to see what these mean. The profilehooks package has two more decorators. The first one we will look at is called **timecall**. It gives us the course run time of the function:

```
# profhooks2.py
from profilehooks import timecall

@timecall
def mem_func():
    lots_of_numbers = list(range(1500))
    x = ['letters'] * (5 ** 10)
    del lots_of_numbers
    return None

if __name__ == '__main__':
    mem_func()
```

When you run this piece of code, you will see something similar to the following output:

```
mem_func (c:\path_to_script\profhooks2.py:3):
  0.141 seconds
```

All decorator does is time the execution time of the function but without the overhead of profiling. It's kind of like using **timeit**.

The last decorator that profhooks provides is called **coverage**. It is supposed to print out the line coverage of a single function. I didn't really find this one all that useful myself, but you're welcome to give it a try on your own.

Finally I just want to mention that you can also run profilehooks on the command line using Python's -m flag:

```
python -m profilehooks mymodule.py
```

The profilehooks package is pretty new, but I think it has some potential.

Wrapping Up

We covered a lot of information in this chapter. You learned how to use Python's built-in modules, timeit and cProfile to time and profile your code, respectively. You also learned how to write your own timing code and use it as a decorator or a context manager. Then we moved on and looked at some 3rd party packages; namely **line_profiler**, **memory_profiler** and **profilehooks**. At this point, you should be well on your way to benchmarking your own code. Give it a try and see if you can find any bottlenecks of your own.