chrono Documentation

Release 1.0.2

Daniel Lindsley

December 17, 2013

Contents

A (BSD licensed) context manager for timing execution. Useful for benchmarking everyday Python code easily/cleanly.

CHAPTER 1

Installation

```
$ pip install chrono
```

CHAPTER 2

Contents

2.1 chrono Tutorial

`chrono` is a pretty small & simple library. It leverages the context manager support added in Python 2.6+ (backported to Python 2.5 via `__future__`). It also works on Python 3.3+ (all examples below are written using Py3).

It provides a simple context manager that you wrap around the code you want to time.

2.1.1 Getting Started

For example, let's say you want to time your awesome Fibonacci function & you want to make sure it's efficient. Your code might look something like...:

```
def fib(n):
    if n in (0, 1):
        return n
    else:
        return fib(n - 1) + fib(n - 2)
```

To time it using `chrono`, you'd call it in the following way...:

```
from chrono import Timer

with Timer() as timed:
    print(fib(10))

print("Seconds taken: {0}".format(timed.elapsed))
```

If everything's setup right, you should get something like the following as output:

```
55
Seconds taken: 0.000102996826172
```

Unfortunately, unless you pass a very large number to `fib`, the time taken is going to be very low & you're not going to get a good sample. If only we could run that function with small values but do it a bunch of times...

2.1.2 Looping

This is where the context manager approach shines. Rather than having to write a wrapper function or similar (like you have to do with the `timeit` module), you simply need to alter your benchmarking code to use a loop...:

```
from chrono import Timer

with Timer() as timed:
    for i in range(100):
        print(fib(10))

print("Seconds taken: {0}".format(timed.elapsed))
```

Now I get this as output:

```
# Lots of "55"s, then...
Seconds taken: 0.0073549747467
```

Much better. Now things like CPU context switches & kernel scheduling will have less of an impact on our benchmark. It's still not perfect (things like averages would help some), but it's an improvement & it was easy.

2.1.3 `Timer`

When invoked with the `with Timer() as ...:` statement, you get back the `Timer` instance. It's a simple object with a couple of useful properties.

‘‘ elapsed‘‘ After the context manager is complete, the time differential is calculated & stored for later reference. This is a floating point number in seconds.

`start` The time (in a floating point Unix timestamp) of when the context manager started.

`end` The time (in a floating point Unix timestamp) of when the context manager completed.

2.2 Running Tests

Setup:

```
$ git clone https://github.com/toastdriven/chrono.git
$ cd chrono
$ virtualenv -p python3 env3
$ . env3/bin/activate
$ pip install nose
```

Running:

```
$ nosetests -s -v tests.py
```

`chrono` is maintained with 100% passing tests at all times.

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*