What is the Python equivalent of Matlab's tic and toc functions?

Asked 13 years, 1 month ago Modified 2 days ago Viewed 242k times

158

What is the Python equivalent of Matlab's tic and toc functions?

python matlab timing

Share Edit Follow Flag

edited Jun 20, 2020 at 9:12
Community Bot
1 1

asked May 1, 2011 at 16:46
Alex
2,204 3 20 19

10 If you really want the direct equivalent, just call `tic = time.time()` and `toc = time.time()` , then `print toc-tic, 'sec Elapsed'` As folks have said below, though, `timeit` is more robust. – Joe Kington May 1, 2011 at 16:53

2 I seem to get better results using @JoeKington's approach in conjunction with timeit.default_timer(), like this for example: `tic = timeit.default_timer(); (U,S,V) = np.linalg.svd(A); toc = timeit.default_timer()` , then `print toc-tic` . – littleO Dec 18, 2016 at 23:43

2 The library pytictoc seems most conveinent, syntax is even slightly neater than ttictoc below. pypi.org/project/pytictoc – FlorianH Oct 7, 2019 at 11:43

15 Answers

Sorted by: Highest score (default)

237

Apart from `timeit` which ThiefMaster mentioned, a simple way to do it is just (after importing `time`):

```
t = time.time()
# do stuff
```

```
elapsed = time.time() - t
```

I have a helper class I like to use:

```
class Timer(object):
    def __init__(self, name=None):
        self.name = name

    def __enter__(self):
        self.tstart = time.time()

    def __exit__(self, type, value, traceback):
        if self.name:
            print('[%s]' % self.name,)
        print('Elapsed: %s' % (time.time() - self.tstart))
```

It can be used as a context manager:

```
with Timer('foo_stuff'):
   # do some foo
   # do some stuff
```

Sometimes I find this technique more convenient than `timeit` - it all depends on what you want to measure.

Share Edit Follow Flag

edited Apr 3, 2019 at 13:51
Jonas Adler
10.7k 5 47 74

answered May 1, 2011 at 16:55
Eli Bendersky
269k 90 358 418

30 @eat: I respectfully disagree. People have been using the unix `time` command to measure runtimes of programs for ever, and this method replicates this inside Python code. I see nothing wrong with it, as long as it's the right tool for the job. `timeit` isn't always that, and a profiler is a much more heavyweight solution for most needs – Eli Bendersky May 2, 2011 at 3:27

4 For the last line I would suggest `print 'Elapsed: %.2f seconds % (time.time() - self.tstart)'` . It is hard to understand without the %.2f. Thanks for great idea. – Can Kavaklıoğlu Jun 19, 2013 at 15:01

7 This looks very convenient at first glance, but in practice requires one to indent the code block one wants to time, which can be quite inconvenient depending on the length of the code block and the editor of choice. Still an elegant solution, which behaves correctly in the case of nested use. – Stefan Nov 15, 2013 at 8:44

10 @rysqui - Isn't the *current time* always a larger number than a *previous time*? I would think that `elapsed = time.time() - t` is the form that always yields a positive value. – Scott Smith Nov 7, 2017 at 22:22

3 @ScottSmith hmmm. yeah i have no idea what i was thinking when I wrote that comment 2 years ago. It seems super wrong, and you seem super correct. Not sure what i was thinking. – rysqui Nov 9, 2017 at 1:41

52

I had the same question when I migrated to python from Matlab. With the help of this thread I was able to construct an *exact* analog of the Matlab `tic()` and `toc()` functions. Simply insert the following code at the top of your script.

```
import time

def TicTocGenerator():
    # Generator that returns time differences
    ti = 0           # initial time
    tf = time.time() # final time
    while True:
        ti = tf
        tf = time.time()
        yield tf-ti # returns the time difference

TicToc = TicTocGenerator() # create an instance of the TicTocGen generator

# This will be the main function through which we define both tic() and toc()
def toc(tempBool=True):
    # Prints the time difference yielded by generator instance TicToc
    tempTimeInterval = next(TicToc)
    if tempBool:
        print( "Elapsed time: %f seconds.\n" %tempTimeInterval )

def tic():
    # Records a time in TicToc, marks the beginning of a time interval
    toc(False)
```

That's it! Now we are ready to fully use `tic()` and `toc()` just as in Matlab. For example

```
tic()

time.sleep(5)
```

```
toc() # returns "Elapsed time: 5.00 seconds."
```

Actually, this is more versatile than the built-in Matlab functions. Here, you could create another instance of the `TicTocGenerator` to keep track of multiple operations, or just to time things differently. For instance, while timing a script, we can now time each piece of the script seperately, as well as the entire script. (I will provide a concrete example)

```
TicToc2 = TicTocGenerator() # create another instance of the TicTocGen generator

def toc2(tempBool=True):
    # Prints the time difference yielded by generator instance TicToc2
    tempTimeInterval = next(TicToc2)
    if tempBool:
    print( "Elapsed time 2: %f seconds.\n" %tempTimeInterval )

def tic2():
    # Records a time in TicToc2, marks the beginning of a time interval
    toc2(False)
```

Now you should be able to time two separate things: In the following example, we time the total script and parts of a script separately.

```
tic()

time.sleep(5)

tic2()

time.sleep(3)

toc2() # returns "Elapsed time 2: 5.00 seconds."

toc() # returns "Elapsed time: 8.00 seconds."
```

Actually, you do not even need to use `tic()` each time. If you have a series of commands that you want to time, then you can write

```
tic()

time.sleep(1)
```

```
toc() # returns "Elapsed time: 1.00 seconds."

time.sleep(2)

toc() # returns "Elapsed time: 2.00 seconds."

time.sleep(3)

toc() # returns "Elapsed time: 3.00 seconds."

# and so on...
```

I hope that this is helpful.

Share Edit Follow Flag

answered Nov 2, 2014 at 2:44
Benben
621 5 5

1 "this is more versatile than the built-in Matlab functions" — I suggest you read the MATLAB docs, you'll see your solution is not nearly as convenient when measuring multiple things at once. And of course this is not an exact analog. – Cris Luengo Nov 27, 2022 at 15:08

25

The absolute best analog of tic and toc would be to simply define them in python.

```
def tic():
    #Homemade version of matlab tic and toc functions
    import time
    global startTime_for_tictoc
    startTime_for_tictoc = time.time()

def toc():
    import time
    if 'startTime_for_tictoc' in globals():
        print "Elapsed time is " + str(time.time() - startTime_for_tictoc) + "
seconds."
    else:
        print "Toc: start time not set"
```

Then you can use them as:

```
tic()
# do stuff
toc()
```

Share Edit Follow Flag

answered Sep 19, 2013 at 19:12

GuestPoster
259 3 2

6 This will not behave correctly in the case of nested use of `tic` and `toc`, which Matlab supports. A little more sophistication would be required. – Stefan Nov 15, 2013 at 8:37

2 I have implemented similar functions in my own code when I needed some basic timing. I would however remove the `import time` outside of both functions, since it can take potentially quite some time. – Bas Swinckels Nov 15, 2013 at 17:19

If you insist on using this technique, and you need it to handle nested tic/toc, make the global a list and let `tic` push to it and `toc` pop from it. – Ahmed Fasih Aug 24, 2016 at 1:26

1 Also I read elsewhere that `timeit.default_timer()` is better than `time.time()` because `time.clock()` might be more appropriate depending on OS – Miguel Feb 14, 2017 at 21:23

@AhmedFasih That's what my answer does, though more things could be improved. – antonimmo Apr 23, 2020 at 7:32

|

20

Usually, IPython's `%time`, `%timeit`, `%prun` and `%lprun` (if one has `line_profiler` installed) satisfy my profiling needs quite well. However, a use case for `tic-toc`-like functionality arose when I tried to profile calculations that were interactively driven, i.e., by the user's mouse motion in a GUI. I felt like spamming `tic`s and `toc`s in the sources while testing interactively would be the fastest way to reveal the bottlenecks. I went with Eli Bendersky's `Timer` class, but wasn't fully happy, since it required me to change the indentation of my code, which can be inconvenient in some editors and confuses the version control system. Moreover, there may be the need to measure the time between points in different functions, which wouldn't work with the `with` statement. After trying lots of Python cleverness, here is the simple solution that I found worked best:

```
from time import time
_tstart_stack = []
```

```
def tic():
    _tstart_stack.append(time())

def toc(fmt="Elapsed: %s s"):
    print fmt % (time() - _tstart_stack.pop())
```

Since this works by pushing the starting times on a stack, it will work correctly for multiple levels of `tic` s and `toc` s. It also allows one to change the format string of the `toc` statement to display additional information, which I liked about Eli's `Timer` class.

For some reason I got concerned with the overhead of a pure Python implementation, so I tested a C extension module as well:

```
#include <Python.h>
#include <mach/mach_time.h>
#define MAXDEPTH 100

uint64_t start[MAXDEPTH];
int lvl=0;

static PyObject* tic(PyObject *self, PyObject *args) {
    start[lvl++] = mach_absolute_time();
    Py_RETURN_NONE;
}

static PyObject* toc(PyObject *self, PyObject *args) {
return PyFloat_FromDouble(
        (double)(mach_absolute_time() - start[--lvl]) / 1000000000L);
}

static PyObject* res(PyObject *self, PyObject *args) {
    return tic(NULL, NULL), toc(NULL, NULL);
}

static PyMethodDef methods[] = {
    {"tic", tic, METH_NOARGS, "Start timer"},
    {"toc", toc, METH_NOARGS, "Stop timer"},
    {"res", res, METH_NOARGS, "Test timer resolution"},
    {NULL, NULL, 0, NULL}
};

PyMODINIT_FUNC
inittictoc(void) {
    Py_InitModule("tictoc", methods);
}
```

This is for MacOSX, and I have omitted code to check if `lvl` is out of bounds for brevity. While `tictoc.res()` yields a resolution of about 50 nanoseconds on my system, I found that the jitter of measuring any Python statement is easily in the microsecond range (and much more when used from IPython). At this point, the overhead of the Python implementation becomes negligible, so that it can be used with the same confidence as the C implementation.

I found that the usefulness of the `tic-toc` -approach is practically limited to code blocks that take more than 10 microseconds to execute. Below that, averaging strategies like in `timeit` are required to get a faithful measurement.

Share Edit Follow Flag

answered Nov 15, 2013 at 17:07
Stefan
4,470 2 31 34

17

You can use `tic` and `toc` from `ttictoc` . Install it with

```
pip install ttictoc
```

And just import them in your script as follow

```
from ttictoc import tic,toc
tic()
# Some code
print(toc())
```

Share Edit Follow Flag

edited Apr 21, 2020 at 18:57

answered Jul 26, 2018 at 21:41
H. Sánchez
576 6 15

I have just created a module [tictoc.py] for achieving nested tic tocs, which is what Matlab does.

8

```
from time import time

tics = []

def tic():
    tics.append(time())

def toc():
    if len(tics)==0:
        return None
    else:
        return time()-tics.pop()
```

And it works this way:

```
from tictoc import tic, toc

# This keeps track of the whole process
tic()

# Timing a small portion of code (maybe a loop)
tic()

# -- Nested code here --

# End
toc()  # This returns the elapse time (in seconds) since the last invocation of
tic()
toc()  # This does the same for the first tic()
```

I hope it helps.

Share Edit Follow Flag

answered May 18, 2017 at 18:52

antonimmo
403 5 8

Nice replication of tic/toc from MATLAB! – Matt May 23, 2017 at 15:50

1 I must warn you that this might not behave as desired when used simultaneously by more than 1 module, since (AFAIK) modules behave like singletons. – antonimmo Jun 27, 2017 at 18:04

1 This is not how MATLAB's tic/top works. There is no stack. You can call `toc` repeatedly to get the time from the same last `tic`. Nested timings work through an output argument to `tic`, which you then pass in to `toc`. – Cris Luengo Nov 27, 2022 at 15:16

5

```
pip install easy-tictoc
```

In the code:

```
from tictoc import tic, toc

tic()

#Some code

toc()
```

Disclaimer: I'm the author of this library.

Share Edit Follow Flag

edited Dec 10, 2019 at 17:41
SecretAgentMan 2,854 7 22 42

answered Dec 10, 2019 at 16:30
Shakib Rahimi 51 1 5

Please don't simply copy another answer, even if it is your own. stackoverflow.com/a/59271862/8239061 – SecretAgentMan Dec 10, 2019 at 17:42

5

Have a look at the `timeit` module. It's not really equivalent but if the code you want to time is inside a function you can easily use it.

Share Edit Follow Flag

answered May 1, 2011 at 16:48
ThiefMaster
316k 84 599 642

Yes, `timeit` is best for benchmarks. It doesn't even have to be a single function, you can pass abritarily complex statements. – user395760 May 1, 2011 at 16:49

10 Well, passing code that is not an extremely simple function call as a string is very ugly. – ThiefMaster May 1, 2011 at 16:50

It also work with lambdas like this (straight from the documentation): `timeit.timeit(lambda: "-".join(map(str, range(100))), number=10000)` – Erik Sillén Jul 27, 2021 at 7:27

4

Updating Eli's answer to Python 3:

```
class Timer(object):
    def __init__(self, name=None, filename=None):
        self.name = name
        self.filename = filename

    def __enter__(self):
        self.tstart = time.time()

    def __exit__(self, type, value, traceback):
        message = 'Elapsed: %.2f seconds' % (time.time() - self.tstart)
        if self.name:
            message = '[%s] ' % self.name + message
        print(message)
        if self.filename:
            with open(self.filename,'a') as file:
                print(str(datetime.datetime.now())+": ",message,file=file)
```

Just like Eli's, it can be used as a context manager:

```
import time
with Timer('Count'):
    for i in range(0,10_000_000):
        pass
```

Output:

```
[Count] Elapsed: 0.27 seconds
```

I have also updated it to print the units of time reported (seconds) and trim the number of digits as suggested by Can, and with the option of also appending to a log file. You must import datetime to use the logging feature:

```
import time
import datetime
with Timer('Count', 'log.txt'):
    for i in range(0,10_000_000):
        pass
```

Share Edit Follow Flag

edited Jul 26, 2018 at 21:04

answered Jun 20, 2018 at 22:05

Josiah Yoder

3,544 4 44 63

3

Building on Stefan and antonimmo's answers, I ended up putting

```
def Tictoc():
    start_stack = []
    start_named = {}

    def tic(name=None):
        if name is None:
            start_stack.append(time())
        else:
            start_named[name] = time()

    def toc(name=None):
        if name is None:
            start = start_stack.pop()
        else:
            start = start_named.pop(name)
            elapsed = time() - start
            return elapsed
        return tic, toc
```

in a `utils.py` module, and I use it with a

```
from utils import Tictoc
tic, toc = Tictoc()
```

This way

- you can simply use `tic()`, `toc()` and nest them like in Matlab
- alternatively, you can name them: `tic(1)`, `toc(1)` or `tic('very-important-block')`, `toc('very-important-block')` and timers with different names won't interfere
- importing them this way prevents interference between modules using it.

(here toc does not print the elapsed time, but returns it.)

Share Edit Follow Flag

answered Oct 22, 2017 at 13:30
Maxim
7,437 1 30 30

1 MATLAB's tic/toc don't nest this way, there'a no stack. I do like the idea of named timers. – Cris Luengo Nov 27, 2022 at 15:18

1

This can also be done using a wrapper. Very general way of keeping time.

The wrapper in this example code wraps any function and prints the amount of time needed to execute the function:

```
def timethis(f):
    import time

    def wrapped(*args, **kwargs):
        start = time.time()
        r = f(*args, **kwargs)
        print "Executing {0} took {1} seconds".format(f.func_name,  time.time()-
start)
        return r
    return wrapped

@timethis
def thistakestime():
    for x in range(10000000):
        pass

thistakestime()
```

Share Edit Follow Flag

edited Jul 27, 2016 at 13:30

answered Jul 27, 2016 at 13:18

Coen Jonker
451 4 5

The wrapper function, timethis is called a decorator. A bit more detailed explanation, here: medium.com/pythonhive/... – Mircea Jan 29, 2018 at 14:44

1

I changed @Eli Bendersky's answer a little bit to use the ctor `__init__()` and dtor `__del__()` to do the timing, so that it can be used more conveniently without indenting the original code:

```
class Timer(object):
    def __init__(self, name=None):
        self.name = name
        self.tstart = time.time()

    def __del__(self):
        if self.name:
            print '%s elapsed: %.2fs' % (self.name, time.time() - self.tstart)
        else:
            print 'Elapsed: %.2fs' % (time.time() - self.tstart)
```

To use, simple put Timer("blahblah") at the beginning of some local scope. Elapsed time will be printed at the end of the scope:

```
for i in xrange(5):
    timer = Timer("eigh()")
```

```
        x = numpy.random.random((4000,4000));
        x = (x+x.T)/2
        numpy.linalg.eigh(x)
        print i+1
    timer = None
```

It prints out:

```
1
eigh() elapsed: 10.13s
2
eigh() elapsed: 9.74s
3
eigh() elapsed: 10.70s
4
eigh() elapsed: 10.25s
5
eigh() elapsed: 11.28s
```

Share Edit Follow Flag

edited Aug 30, 2017 at 5:37

answered May 27, 2015 at 13:09

Shaohua Li
399 3 9

3 An issue with this implementation is the fact, that `timer` is not deleted after the last call, if any other code follows after the `for` loop. To get the last timer value, one should delete or overwrite the `timer` after the `for` loop, e.g. via `timer = None`. – bastelflp Jul 5, 2016 at 15:41

1 @bastelflp Just realized that I misunderstood what you meant... Your suggestion has been incorporated in the code now. Thanks. – Shaohua Li Aug 30, 2017 at 5:38

0

Based on the answer of @Maxim, here is a simplified solution:

```
def tictoc():
    """Python implementation of MATLAB tic-toc functionality."""
    from time import perf_counter_ns
    hashmap = {}
    def tic(key: str=None) -> None:
        """named tic method"""
        hashmap[key] = perf_counter_ns()
```

```
            return None
        def toc(key: str=None) -> float:
            """named toc method"""
            initial_ns = hashmap[key]
            current_ns = perf_counter_ns()
            elapsed_ns = current_ns - initial_ns
            elapsed_s  = elapsed_ns / (10**9)  # convert ns to s
            print(f"Elapsed time is {elapsed_s} seconds.")
            return elapsed_s
        return tic, toc
```

Assuming it is stored in a file called `utils.py`, it is used in the following manner:

```
from utils import tictoc

tic, toc = tictoc()

tic()
toc()
toc()
tic()
toc()

tic('foo')
toc('foo')
toc('foo')

toc()
```

Unlike the original, this version does not crash when you run `toc()` multiple times and properly resets when running `tic` while retaining the key-value timer functionality. It also removes an unnecessary list/array data structure and multiple conditional branches in the code, which induce slowdowns. Hashmap lookup times are constant at O^1, which is faster than resizing arrays. Also, I have switched to the `perf_counter_ns` timer, which is based on `perf_counter` and avoids the precision loss caused by the `float` type.

If anyone wants to go lower level, here is a version using the Python C API:

```
// tictoc.c
#define PY_SSIZE_T_CLEAN  // REQUIRED
#include <Python.h>       // Python C API

#include <stdint.h>       // uint64_t
```

```
#include <time.h>         // timespec
#include <sys/time.h>     // time types
#include <stdio.h>        // printf

typedef struct {
    uint64_t initial_ns;
    uint64_t current_ns;
    uint64_t elapsed_ns;
    double elapsed_s;
} timer;

timer t;

struct timespec ts;  // C11

// Unix POSIX clock time scaled to microseconds (us; 10^6 sec)
uint64_t posix_clock_time () {
    timespec_get(&ts, TIME_UTC);
    return ts.tv_sec * 1000000000 + ts.tv_nsec;  // ns = 10^-9 s
}

// tic
static PyObject* tic(PyObject* self, PyObject* args) {
    t.initial_ns = posix_clock_time();
    Py_RETURN_NONE;
}

// toc
static PyObject* toc(PyObject* self, PyObject* args) {
    t.current_ns = posix_clock_time();
    t.elapsed_ns = t.current_ns - t.initial_ns;
    t.elapsed_s  = (double)t.elapsed_ns / (double)1000000000.0;  // s = 10^9 ns
    printf("Elapsed time is %f seconds.\n", t.elapsed_s);
    return PyFloat_FromDouble(t.elapsed_s);
}

static PyMethodDef methods[] = {
    {"tic", tic, METH_NOARGS, "Start timer"},
    {"toc", toc, METH_NOARGS, "Stop timer"},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef tictoc = {
    PyModuleDef_HEAD_INIT,
    "tictoc",    // name of module
    "A timer function analogous to MATLAB's tic-toc.",  // module documentation,
may be NULL
```

```
    -1,          // size of per-interpreter state of module; -1 if module keeps
state in global variables
    methods
};

PyMODINIT_FUNC PyInit_tictoc(void) {
    return PyModule_Create(&tictoc);
}
```

Install it with a `setup.py` file:

```
# setup.py
from distutils.core import setup, Extension

module1 = Extension(
    'tictoc',
    sources = ['tictoc.c']
)

setup(name = 'tictoc',
    version = '1.0',
    description = 'The tictoc package',
    ext_modules = [module1]
)
```

From the command line, run `python setup.py build` and Bob's your uncle. This compiles to a shared library that provides basic tic-toc functionality similar to MATLAB:

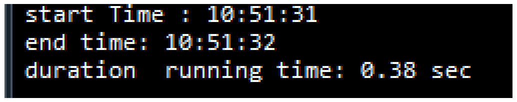
```
from tictoc import tic, toc

tic()
toc()
toc()

tic()
toc()
toc()
```

Share Edit Follow Flag edited Mar 15, 2023 at 22:09 answered Mar 15, 2023 at 18:21

Adam Erickson
6,233 2 47 35

0

To show the duration time of running a code in Python, you can use the `time` module to calculate the elapsed time between the start and end of the code execution. Here's an example:



```
start Time : 10:51:31
end time: 10:51:32
duration  running time: 0.38 sec
```

```
import time
start_time = time.time()
print("start Time : {}".format(time.strftime("%H:%M:%S",
time.localtime(start_time))))

# Run code here

end_time = time.time()
print("end time: {}".format(time.strftime("%H:%M:%S", time.localtime(end_time))))
duration = end_time - start_time
print("duration  running time: {:.2f} sec".format(duration))
```

Share Edit Follow Flag

answered 2 days ago
hamed aghapanah
19 1

0

This solution works for my profiling needs:

```
from time import time
import inspect

def tic():
    tic.t = time()

def toc(message=None):
    time_elapsed = time() - tic.t
    if message is None:
        message = inspect.currentframe().f_back.f_lineno
        print(message, time_elapsed)
        tic.t = time()
        return time_elapsed
```

Then you can just paste a lot of `toc()` s in your code, and you have a pretty powerful profiler. (message defaults to the caller's code line in file)

Share Edit Follow Flag

answered Nov 27, 2022 at 11:20

Michael Doron

85 7