Python programming — Profiling

Finn Årup Nielsen

DTU Compute
Technical University of Denmark

January 3, 2014

Overview

`time` and `timeit`

profiling

Simpel profiling with `time`

`time` and `timeit` modules and `times()` function in the `os` module (Langtangen, 2005, p. 422+)

Example with Python Standard Library's `time` and its `clock` function:

```
from numpy import array, dot
import time
elapsed = time.time()
cpu = time.clock()
dummy = dot(array((100000,1)), array((100000,1)))
print time.time() - elapsed
print time.clock() - cpu
```

Profiling with `timeit`

Python Standard Library's `timeit` can loop a piece of code, executing it many times and measuring the overall timing.

Useful for code that is fast.

Python profiling with “profile”

Profiling: Measuring of time of execution, number of function calls, etc.

```
python -m cProfile -s time mypythonprogram
```

Try/except vs. testing for dict element

“It’s Easier to Ask Forgiveness Than Permission” (EAFP) vs. “Look Before You Leap” (LBYL)

Consider a dictionary where you want to get an element where the key does not necessarily exists. The following will result in an KeyError exception

```
>>> d = {'a': 1}
>>> d['b']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'b'
```

Is it faster/better to encapsulate the code in a try/except block or test for the existence first?

. . . Try/except vs. testing for dict element

It’s Easier to Ask Forgiveness Than Permission” (EAFP): Using try/except

```
d = {'a': 1}
keys = ['a', 'b']
for key in keys:
    try:
        d[key] += 3
    except KeyError:
        d[key] = 1
```

In this case the except block is executed if the key (here `"b"`) does not exist:

```
>>> d
{'a': 4, 'b': 1}
```

. . . Try/except vs. testing for dict element

“Look Before You Leap” (LBYL): check if element exist

```
d = {’a’: 1}
keys = [’a’, ’b’]
for key in keys:
    if key in d:
        d[key] += 3
    else:
        d[key] = 1
```

. . . Try/except vs. testing for dict element

See are profiling of the cases made by Patrick Altman in Try / Except Performance in Python: A Simple Test

Generally, in standard Python an exception is expensive (but a try is not).

The if condition testing also requires some time to execute.

https://gist.github.com/fnielsen/6127124

. . . Try/except vs. testing for dict element

Note in this simple case we could (also) have used defaultdict from the collections module:

```
import collections
d = collections.defaultdict(lambda: -2)
d['a'] = 1
keys = ['a', 'b']
for key in keys:
    d[key] += 3
```

Or the `setdefault` method in the standard `dict` class:

```
d = {'a': 1}
keys = ['a', 'b']
for key in keys:
    d[key] = d.setdefault(key, -2) + 3
```

Graphical representation

Graphical representation of call graph

pycallgraph

(Debian package: `python-pycallgraph`)

More information

Huy Nguyen, A guide to analyzing Python performance: Unix `time`, timing context manager, `line_profiler`, `memory_profiler` and `objgraph`.

Summary

Do not write inefficient code, unless there are reason for it: readability, maintainability, . . .

Profile you code.

For short code snippets use `timeit`.

For overall profiling use `cProfile`. This facility is directly available in, e.g., Spyder IDE.

Consider other tools: memory profiling, pycallgraph and objgraph.

References

Langtangen, H. P. (2005). *Python Scripting for Computational Science*, volume 3 of *Texts in Computational Science and Engineering*. Springer. ISBN 3540294155.