How to Write Faster Python

Matt Belisle

March 19, 2013

Contents

1 Introduction

This article is a summary of what Matt learned at PyCon 2013 about how to make Python programs faster. A digital version of this article can be found here:

`https://github.com/matthewbelisle-wf/faster_python/blob/master/faster_python.pdf`

In order to get the most out of this article, open up an IPython prompt yourself and play around with the examples below. Sections 2 and 3 can make a slow program orders of magnitude faster. Sections 4 and 5 can make a slow program fractionally faster.

2 Use Built-in Types and Functions

This seems obvious but worth mentioning. Built-in data structures like list, dict, complex, etc. have two advantages compared to whatever we write.

1. They were written in C.
2. They were written by Guido.

Before writing a class or a function make sure there isn't one already built into Python. There are a lot of built in functions that most of us don't know about.

3 Use the Appropriate Data Type

A common mistake for Python programmers is to use a list when we should be using a tuple, set, deque, or a generator instead.

- List
  `http://docs.python.org/2/library/functions.html#list`
- Tuple
  `http://docs.python.org/2/library/functions.html#tuple`
- Set
  `http://docs.python.org/2/library/functions.html#set`
- Deque
  `http://docs.python.org/2/library/collections.html#collections.deque`
- Generator
  `http://docs.python.org/2/reference/simple_stmts.html#yield`

3.1 Set vs List

For inclusion testing, use a set which is $O(1)$ time complexity instead of a list which is $O(n)$.

```
$ ipython
In [1]: items = list(xrange(10 ** 8))

In [2]: timeit None in items
1 loops, best of 3: 3.04 s per loop

In [3]: items = set(xrange(10 ** 8))

In [4]: timeit None in items
10000000 loops, best of 3: 63.2 ns per loop
```

3.2 Deque vs List

For prepending/appending, use a deque which is $O(1)$ time complexity instead of a list which is $O(n)$.

```
$ ipython
In [1]: items = list(xrange(10 ** 7))

In [2]: timeit items.insert(0, None)
100 loops, best of 3: 8.04 ms per loop

In [3]: from collections import deque

In [4]: items = deque(xrange(10 ** 7))

In [5]: timeit items.appendleft(None)
10000000 loops, best of 3: 104 ns per loop
```

3.3 Tuple vs List

Use a tuple when it's ok for the object to be immutable since a tuple takes up less memory.

```
$ ipython
In [1]: import sys
```

```
In [2]: sys.getsizeof([100, 200])
Out[2]: 88

In [3]: sys.getsizeof((100, 200))
Out[3]: 72
```

3.4 Generator vs List

For iterables where you only need to loop through one variable at a time you should use a generator like xrange(), which is $O(1)$ space complexity instead of a list like range() which is $O(n)$ space complexity.

```
$ ipython
In [1]: import sys

In [2]: sum(range(1000))
Out[2]: 499500

In [3]: sys.getsizeof(range(1000))
Out[3]: 8072

In [4]: sum(xrange(1000))
Out[4]: 499500

In [5]: sys.getsizeof(xrange(1000))
Out[5]: 40
```

3.5 Quiz

Instead of a list, which data type would make this intersect call 200 times faster?[1]

```
$ ipython
In [1]: def intersect(list1, list2):
   ...:     return [i for i in list1 if i in list2]
   ...:

In [2]: import random

In [3]: list1 = random.sample(xrange(10000), 1000)
```

[1] Answer: A set. Extra credit if you noticed that using the built-in intersect() method would make it 800 times faster.

```
In [4]: list2 = random.sample(xrange(10000), 1000)

In [5]: len(intersect(list1, list2))
Out[5]: 109
```

4 Use Attribute Lookups Sparingly

Accessing an attribute on an object performs a hash table lookup behind the scenes, so it is best to avoid using attributes inside a loop.

```
$ ipython
In [1]: import math

In [2]: timeit math.sqrt(100)
10000000 loops, best of 3: 147 ns per loop

In [3]: sqrt = math.sqrt

In [4]: timeit sqrt(100)
10000000 loops, best of 3: 111 ns per loop
```

5 Use Local Scope when Possible

When the python interpreter does a variable lookup, it searches the local scope first. If it can't find it there, it searches the global scope, and if it can't find it there it searches the built-in scope before throwing a NameError.

`http://docs.python.org/2/tutorial/classes.html#python-scopes-and-namespaces`

This means that local variable lookups are a little faster than global variable lookups.

```
$ ipython
In [1]: GLOBAL_VAR = 10

In [2]: def test_global():
   ...:     for i in xrange(10 ** 7):
   ...:         GLOBAL_VAR
   ...:
```

```
In [3]: def test_local():
   ...:     local_var = 10
   ...:     for i in xrange(10 ** 7):
   ...:         local_var
   ...:

In [4]: timeit test_global()
1 loops, best of 3: 358 ms per loop

In [5]: timeit test_local()
1 loops, best of 3: 298 ms per loop
```

6 Test for Hotspots

There are several tools available for locating inefficient parts of a program. Use whichever one you like best.

6.1 Time Complexity

- timeit - Times one line snippets
  `http://docs.python.org/2/library/timeit.html`
- cProfile - Collects stats about time spent in each function
  `http://docs.python.org/2/library/profile.html#module-cProfile`
- kernprof.py - Collects stats about time spent on each line
  `https://pypi.python.org/pypi/line_profiler`

6.2 Memory Complexity

- heapy - Takes memory measuremeants inside Python on any OS
  `https://pypi.python.org/pypi/guppy/`
- valgrind - Takes memory measurements for any program running on Unix
  `http://valgrind.org/`