WRITING FASTER CODE

WRITING FASTER CODE AND NOT HATING YOUR JOB AS A SOFTWARE DEVELOPER

WRITING FASTER PYTHON

Sebastian Witowski

PYTHON WAS NOT MADE TO BE FAST...

...BUT TO MAKE DEVELOPERS FAST.

> *It was nice to learn Python;*
> *a nice afternoon*

Donald Knuth

Would you like your FIRST program EVER to be like:

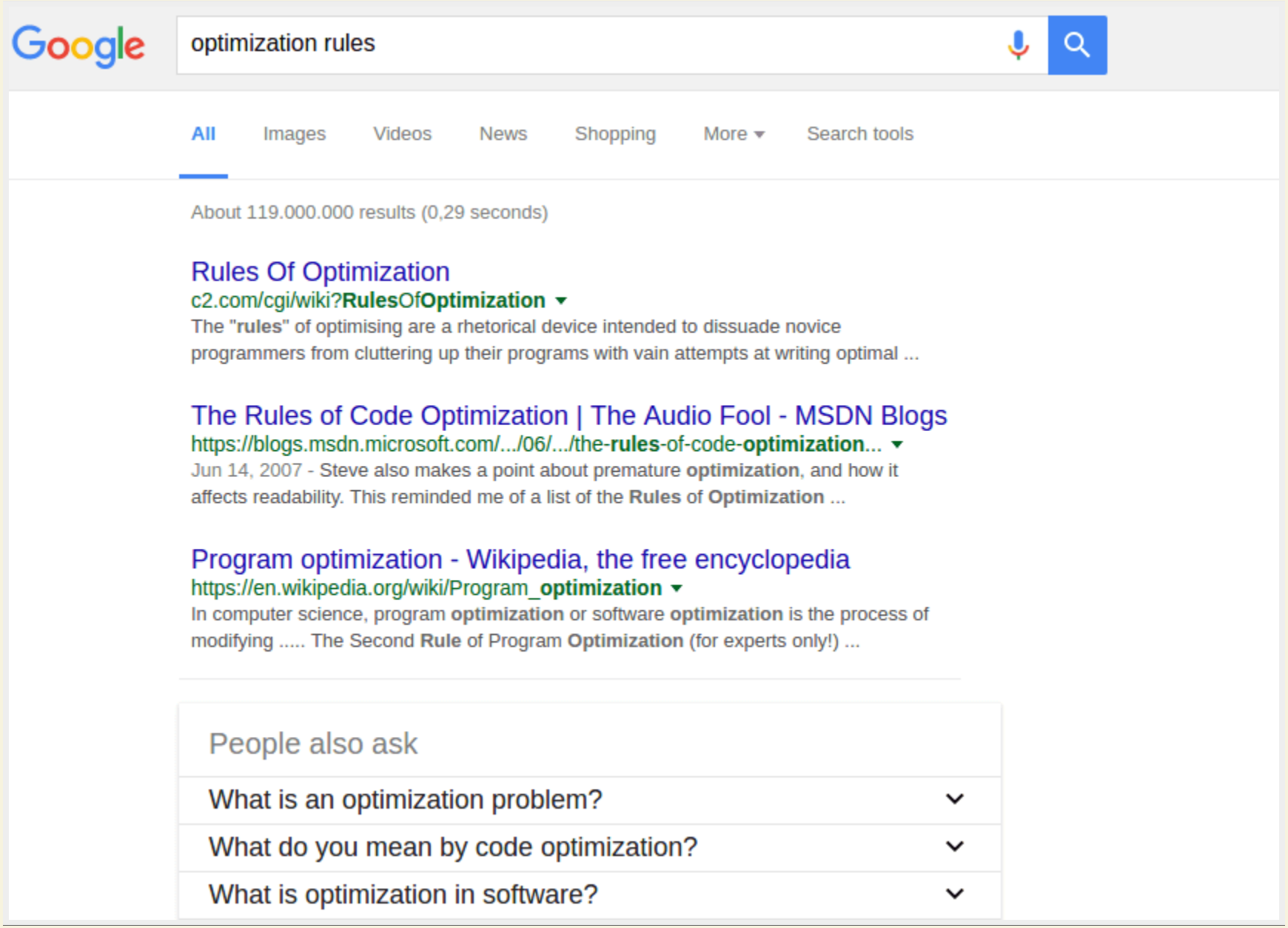
```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

or

```
print("Hello, world!")
```

Source: https://www.shoop.io/en/blog/25-of-the-most-popular-python-and-django-websites

OPTIMIZATION


Google
optimization rules
All
Images
Videos
News
Shopping
More
Search tools
About 119.000.000 results (0,29 seconds)
Rules Of Optimization
c2.com/cgi/wiki?RulesOfOptimization
The "rules" of optimising are a rhetorical device intended to dissuade novice programmers from cluttering up their programs with vain attempts at writing optimal ...
The Rules of Code Optimization | The Audio Fool - MSDN Blogs
https://blogs.msdn.microsoft.com/.../06/.../the-rules-of-code-optimization...
Jun 14, 2007 - Steve also makes a point about premature optimization, and how it affects readability. This reminded me of a list of the Rules of Optimization ...
Program optimization - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/Program_optimization
In computer science, program optimization or software optimization is the process of modifying ..... The Second Rule of Program Optimization (for experts only!) ...
People also ask
What is an optimization problem?
What do you mean by code optimization?
What is optimization in software?

Rules Of Optimization

The "rules" of optimising are a rhetorical device intended to dissuade novice programmers from cluttering up their programs with vain attempts at writing optimal code. They are:

1. **FirstRuleOfOptimization - Don't.**
2. **SecondRuleOfOptimization - Don't... *yet.***
3. **ProfileBeforeOptimizing**

It is uncertain at present, whether cute devices such as this have, or ever will, change any attitudes.

It changed mine.

Mine, too.

Source:

MichaelJackson used to say (when asked about optimization):

1. Don't.
2. Don't Yet (for experts only).

This is republished in JonBentley's ProgrammingPearls.

And lets not forget these famous quotes:

> "The best is the enemy of the good."
>
> -- MrVoltaire
>
> "More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason - including blind stupidity."
>
> -- W.A. Wulf
>
> "We should forget about small efficiencies, say about 97% of the time: PrematureOptimization is the root of all evil."
>
> -- DonKnuth (who attributed the observation to CarHoare)

See: OptimizeLater, LazyOptimization, OptimizationUnitTest, OptimizationStories, http://c2.com/cgi/wiki?search=optimiz, UniformlySlowCode, CodeDepreciation, RulesOfOptimizationClub

---

CategoryOptimization

View edit of May 6, 2009 or FindPage with title or text search

1. DON'T

2. DON'T... YET

3. PROFILE

- cProfile
- pstats
- RunSnakeRun, SnakeViz

LEVELS OF OPTIMIZATION

- Design
- Algorithms and data structures

```
sum = 0
for x in range(1, N + 1):
    sum += x
print sum
```

```
print N * (1 + N) / 2
```

LEVELS OF OPTIMIZATION

- Design
- Algorithms and data structures
- Source code
- Build level
- Compile level
- Assembly level
- Runtime level

OPTIMIZATION IS ALL ABOUT THE SPEED

... AND MEMORY

... AND DISK SPACE

... DISK I/O

... NETWORK I/O

... POWER CONSUMPTION

... AND MORE.

> *Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live*

John Woods

WRITING FAST PYTHON

A.K.A SOURCE CODE OPTIMIZATION

SETUP

Python 3.5.1 (IPython 1.2.1)

```
def ultimate_answer_to_life():
    return 42
```

```
>>> %timeit ultimate_answer_to_life()
10000000 loops, best of 3: 87.1 ns per loop
```

2.72×10^{21} times faster than in The Hitchhiker's Guide to the Galaxy ;-)

#1 COUNT ELEMENTS IN A LIST

```
how_many = 0
for element in ONE_MILLION_ELEMENTS:
    how_many += 1
print how_many
```

26.5 ms

```
print len(ONE_MILLION_ELEMENTS)
```

96.7 ns

274 000 times faster

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

And collections module.

#2 FILTER A LIST

```
output = []
for element in MILLION_NUMBERS:
    if element % 2:
        output.append(element)
```

222 ms

```
list(filter(lambda x: x % 2, MILLION_NUMBERS))
```

234 ms

```
[item for item in MILLION_NUMBERS if item % 2]
```

127 ms

75% faster

#2.5 LIST WITH INDEX

```
output = {}
for i in range(0, len(MILLION)):
    output[i] = MILLION[i]
```

334 ms

```
output = {}
for i, element in enumerate(MILLION):
    output[i] = element
```

220 ms

```
{i: element for i, element in enumerate(MILLION)}
```

218 ms

50% faster

#3 PERMISSIONS OR FORGIVENESS ?

```
class Foo(object):
    hello = 'world'
foo = Foo()
```

```
if hasattr(foo, 'hello'):
    foo.hello
```

149 ns

```
try:
    foo.hello
except AttributeError:
    pass
```

43.1 ns

3.5 times faster

#3 PERMISSIONS OR FORGIVENESS ?

```
if (hasattr(foo, 'foo') and hasattr(foo, 'bar')
    and hasattr(foo, 'baz')):
    foo.foo
    foo.bar
    foo.baz
```

401 ns

```
try:
    foo.foo
    foo.bar
    foo.baz
except AttributeError:
    pass
```

110 ns

3.64 times faster

#3 PERMISSIONS OR FORGIVENESS ?

```
class Bar(object):
    pass
bar = Bar()
```

```
if hasattr(bar, 'hello'):
    bar.hello
```

428 ns

```
try:
    bar.hello
except AttributeError:
    pass
```

536 ns

25% slower

#4 MEMBERSHIP TESTING

```
def check_number(number):
    for item in MILLION_NUMBERS:
        if item == number:
            return True
    return False
```

```
%timeit check_number(500000)
```

18 ms

```
500000 in MILLION_NUMBERS
```

8.45 ms

2 times faster

#4 MEMBERSHIP TESTING

```
100 in MILLION_NUMBERS
```

1.55 µs

```
999999 in MILLION_NUMBERS
```

15.7 ms

#4 MEMBERSHIP TESTING

```
MILLION_SET = set(MILLION_NUMBERS)
%timeit 100 in MILLION_SET
```

46.3 ns

33 times faster (vs list)

```
%timeit 999999 in MILLION_SET
```

63.3 ns

248 000 times faster (vs list)

```
%timeit set(MILLION_NUMBERS)
```

106 ms

#5 REMOVE DUPLICATES

```
unique = []
for element in MILLION_ELEMENTS:
    if element not in unique:
        unique.append(element)
```

8.29 s

```
set(MILLION_ELEMENTS)
```

19.3 ms

400 times faster

Trick with OrderedDict (if order matters)

#6 LIST SORTING

```
sorted(MILLION_RANDOM_NUMBERS)
```

467 ms

```
MILLION_RANDOM_NUMBERS.sort()
```

77.6 ms

6 times faster

#7 1000 OPERATIONS AND 1 FUNCTION

```
def square(number):
    return number**2
squares = [square(i) for i in range(1000)]
```

399 µs

```
def compute_squares():
    return [i**2 for i in range(1000)]
```

314 µs

27% faster

#8 CHECKING FOR TRUE

```
if variable == True:
```

35.8 ns

```
if variable is True:
```

28.7 ns

24% faster

```
if variable:
```

20.6 ns

73% faster

#8 CHECKING FOR FALSE

```
if variable == False:
```

35.1 ns

```
if variable is False:
```

26.9 ns

30% faster

```
if not variable:
```

19.8 ns

77% faster

#8 CHECKING FOR EMPTY LIST

```
if len(a_list) == 0:
```

91.7 ns

```
if a_list == []:
```

56.3 ns

60% faster

```
if not a_list:
```

32.4 ns

280% faster

#9 DEF VS LAMBDA

```
def greet(name):
    return 'Hello {}!'.format(name)
```

329 ns

```
greet = lambda name: 'Hello {}!'.format(name)
```

332 ns

#9 DEF VS LAMBDA

```
>>> dis.dis(greet)
 0 LOAD_CONST    1 ('Hello {}!')
 3 LOAD_ATTR     0 (format)
 6 LOAD_FAST     0 (name)
 9 CALL_FUNCTION 1 (1 positional, 0 keyword pair)
12 RETURN_VALUE
```

Stack Overflow on when lambda might be necessary

#10 LIST() OR []

```
list()
```

104 ns

```
[]
```

22.5 ns

4.6 times faster

#10 DICT() OR {}

```
dict()
```

161 ns

```
{}
```

93 ns

1.7 times faster

DANGER ZONE

#11 VARIABLES ASSIGNMENT

```
q=1
w=2
e=3
r=4
t=5
y=6
u=7
i=8
o=9
p=0
```

71.8 ns

```
q,w,e,r,t,y,u,i,o,p = 1,2,3,4,5,6,7,8,9,0
```

56.4 ns

27% faster (but please don't)

#12 VARIABLES LOOKUP

```
def squares(MILLION_NUMBERS):
    output = []
    for element in MILLION_NUMBERS:
        output.append(element*element)
    return output
```

149 ms

```
def squares_faster(MILLION_NUMBERS):
    output = []
    append = output.append # <= !!!!!!!!
    for element in MILLION_NUMBERS:
        append(element*element)
    return output
```

110 ms

35% faster (and 27% more confusing)

SUMMARY

- There are different kinds of optimization
- There are different levels of optimization
- Source code optimizations adds up
- Source code optimizations is cheap
  - Idiomatic Python
  - Don't reinvent the wheel
- Profile your code and be curious!

THANK YOU!

HAPPY AND FAST CODING!