一般人不知道的 Python 秘密

孙宇聪

CTO @ Coding

叶雨飞

Google SRE 2007-2015

Coding CTO 2015 – Current

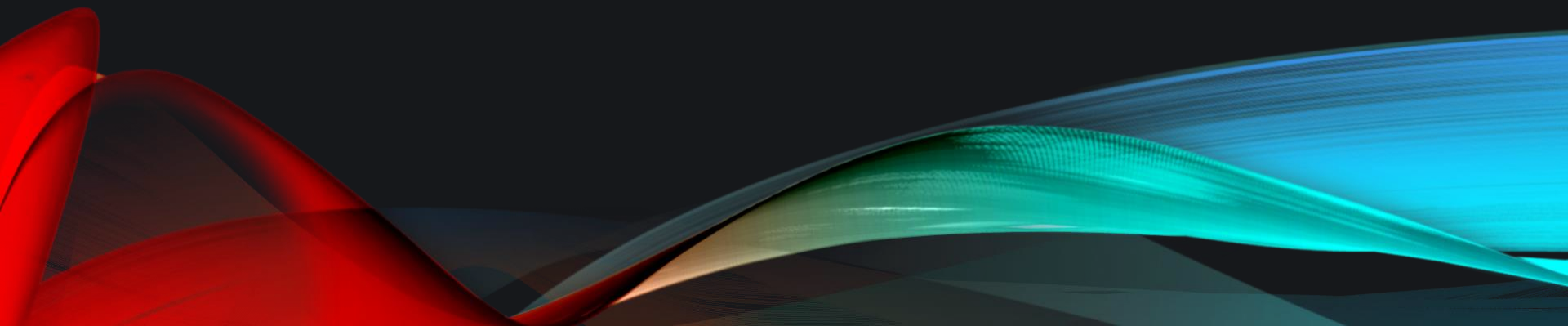
CODING

CLOUD DEVELOPMENT

No python in https://coding.net (yet?)

Me & Python

"Your code is too simple to review" – Guido Van Rossum



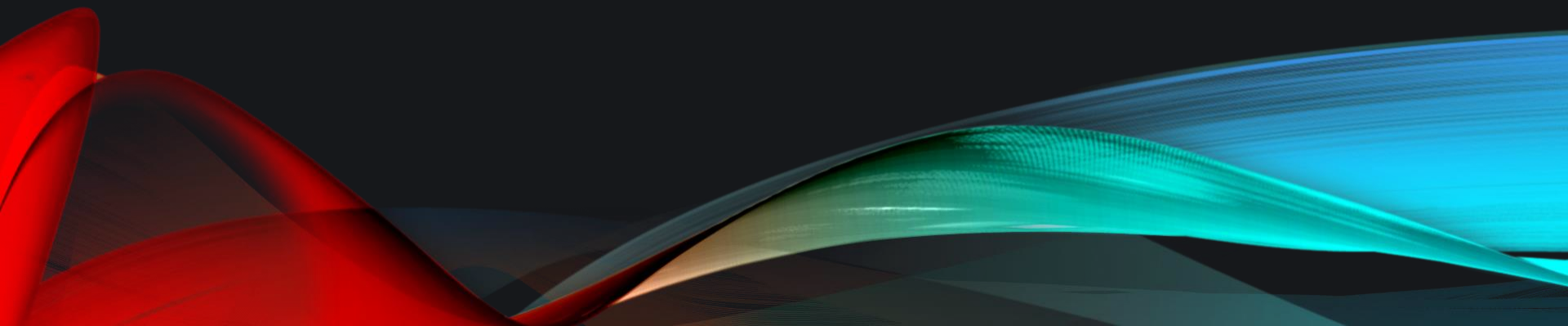
Python Is Great

- Easy, concise language to learn and use.

- My projects
  - A Django based REST backend running on GAE.
  - An decision making backend for machine management tools.
  - Several command line tools
  - Countless scripts.

Python is meh…

Once you learned all the little things



Python is not that great

- Python is interpreted rather than compiled.
- Python is Dynamically Typed rather than Statically Typed.
- Python's object model can lead to inefficient memory access

Not So Great Things

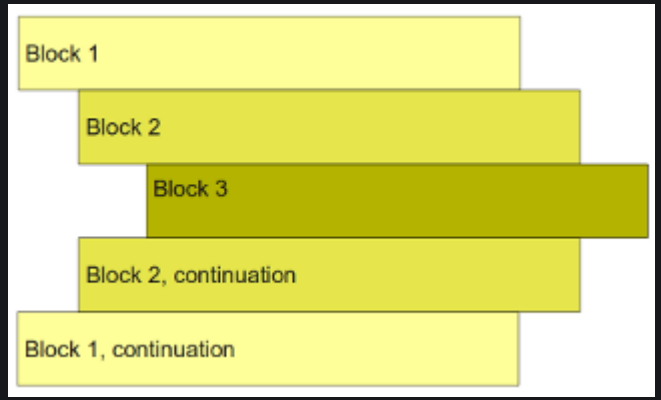
Style

Performance

Maintenance

Style: Whitespace

- Inconsistent display
- Tab/whitespace nightmare
- Extract code into function is huge PITA.
- VCS Merge issue



Numerical

- Everything is a PyObject
- Every operation involves looking up type code
- Small Integer cache (use is to find out)

```c
/* C code */
int a = 1;
int b = 2;
int c = a + b;
```

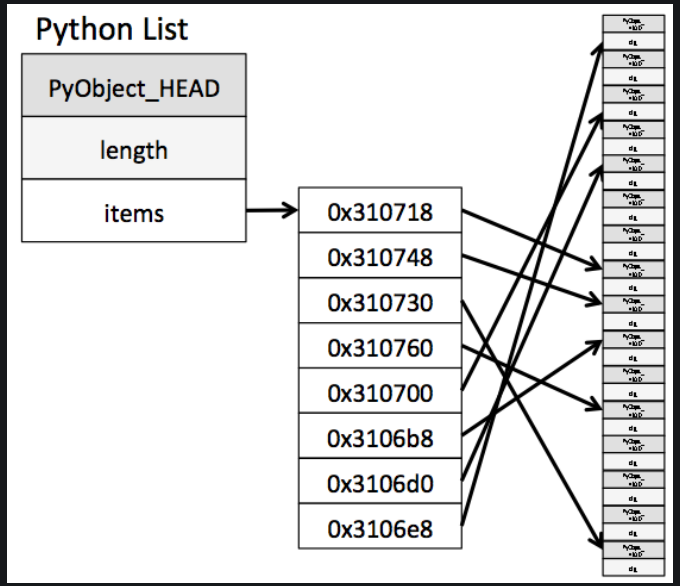
```python
# python code
a = 1
b = 2
c = a + b
```

String

- Remember: String is immutable!
- Avoid +/+=, use Join()
- Unicode is another beast.

List

- Python List is not linked list, it is dynamic array
- Try not insert / delete too much
- Use Deque



Operation	Average Case
Copy	O(n)
Append[1]	O(1)
Insert	O(n)
Get Item	O(1)
Set Item	O(1)
Delete Item	O(n)
Iteration	O(n)

Dictionary

- What is wrong?

- Items() / iteritems()
- Get(x, y)
- Setdefault(x, y)

```
return_dict = {}
for line in file_handle:
        line_list = line.split()
        if line_list[0] in return_dict.keys():
                return_dict[line_list[0]].append(line_list)
        else:
                return_dict[line_list[0]] = [line_list]
```

Tuple

- Tuple is immutable
- But their values can change!
- Tuple is a data structure (namedtuple), use it!

Functions

- Default arguments
- Late binding

```
def append_to(element, to=[]):
    to.append(element)
    return to
```

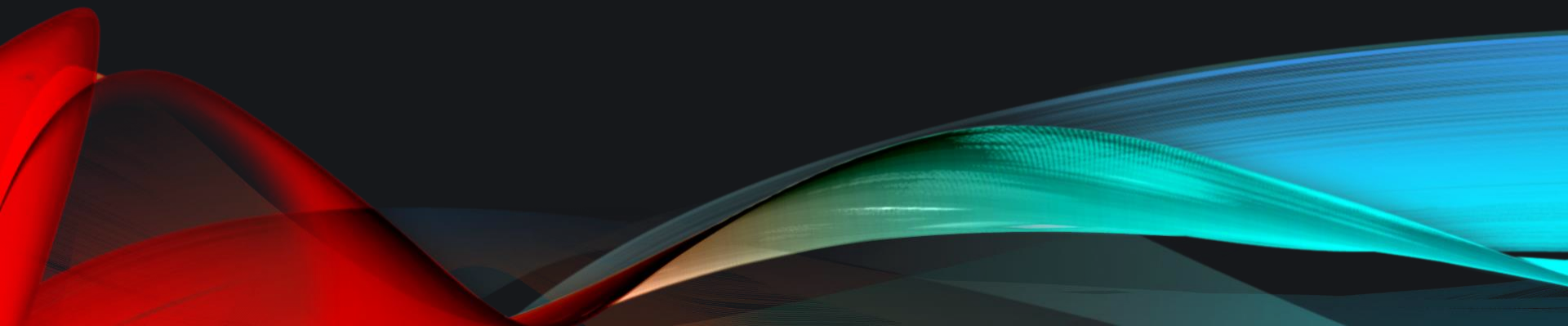
```
my_list = append_to(12)
print my_list

my_other_list = append_to(42)
print my_other_list
```

What Does Happen

```
[12]
[12, 42]
```

Global Interpreter Lock

You know nothing…



Global Interpreter Lock

- Only one thread can run in the interpreter at one time
- Diabolical behavior on multicore machines
- Running in one thread is usually faster.

An Experiment

- Consider this trivial CPU-bound function

```
def countdown(n):
    while n > 0:
        n -= 1
```

- Run it once with a lot of work

```
COUNT = 100000000   # 100 million
countdown(COUNT)
```

- Now, subdivide the work across two threads

```
t1 = Thread(target=countdown,args=(COUNT//2,))
t2 = Thread(target=countdown,args=(COUNT//2,))
t1.start(); t2.start()
t1.join(); t2.join()
```

A Mystery

- Performance on a quad-core MacPro

  Sequential : 7.8s
  Threaded (2 threads) : 15.4s (2X slower!)

- Performance if work divided across 4 threads

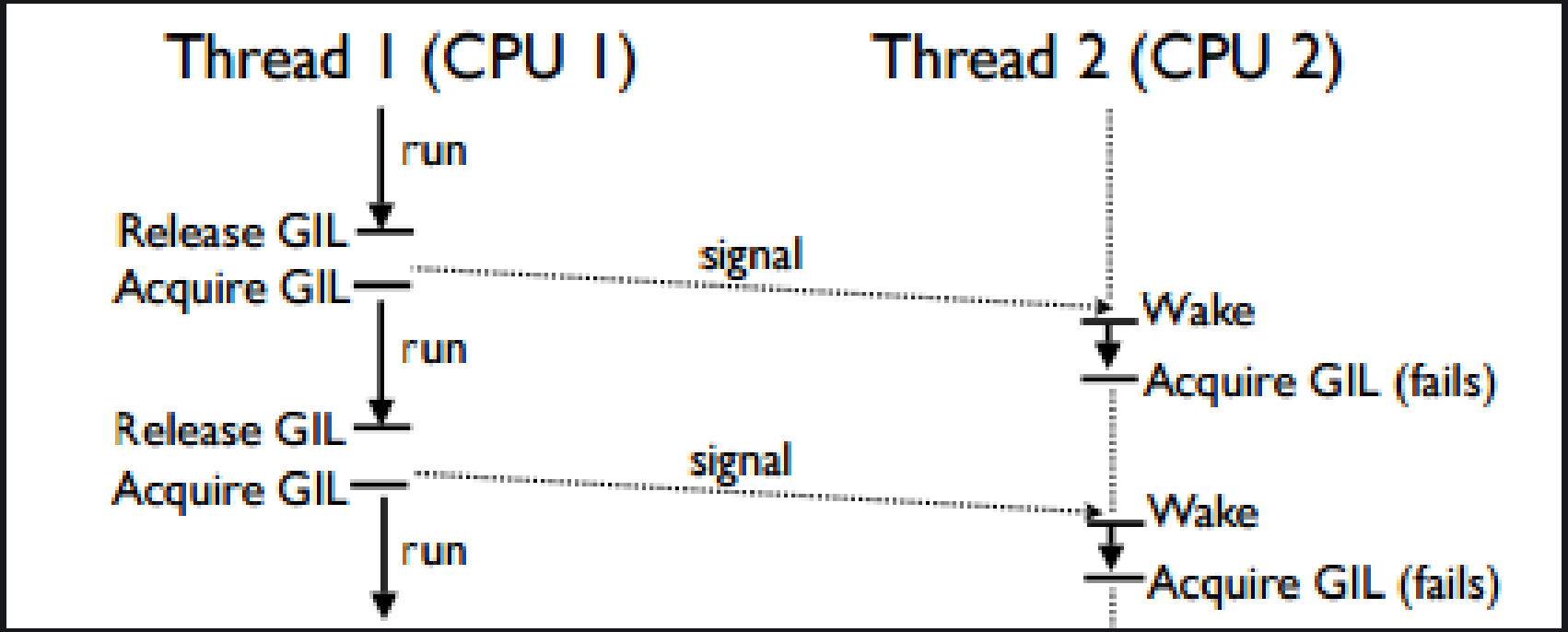
  Threaded (4 threads) : 15.7s (about the same)

- Performance if all but one CPU is disabled

  Threaded (2 threads) : 11.3s (~35% faster than running
  Threaded (4 threads) : 11.6s with all 4 cores)

- Think about it...

Multicore GIL Battle



Life with GIL

- CPU-bound task can only ever use single core.
- Multicore is slower than single core
- I/O-bound task may get starved by CPU-bound one.
- Most of the time, threading is moot.

Use multiprocessing

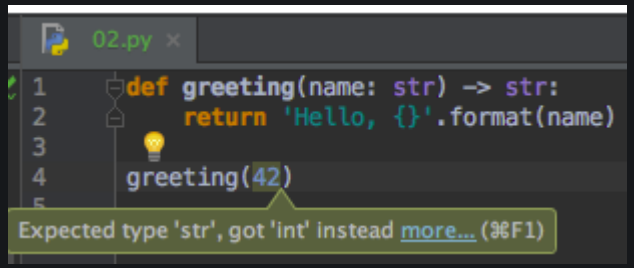
```
from multiprocessing import Pool, TimeoutError
import time
import os

def f(x):
    return x*x

if __name__ == '__main__':
    pool = Pool(processes=4)              # start 4 worker processes

    # print "[0, 1, 4,..., 81]"
    print pool.map(f, range(10))

    # print same numbers in arbitrary order
    for i in pool.imap_unordered(f, range(10)):
        print i

    # evaluate "f(20)" asynchronously
    res = pool.apply_async(f, (20,))      # runs in *only* one process
    print res.get(timeout=1)              # prints "400"
```

Maintenance Issues

- 100% Unit test is not enough.
- Scripting languages allow small things to be written quickly because they are concise.
- Maintenance of large systems is easiest when there is redundancy that allows errors to be caught.
- Optimize for maintenance, not the original author.


CODING
CLOUD DEVELOPMENT

PEP 484


02.py
1 def greeting(name: str) -> str:
2 return 'Hello, {}'.format(name)
3
4 greeting(42)
Expected type 'str', got 'int' instead more... (⌘F1)