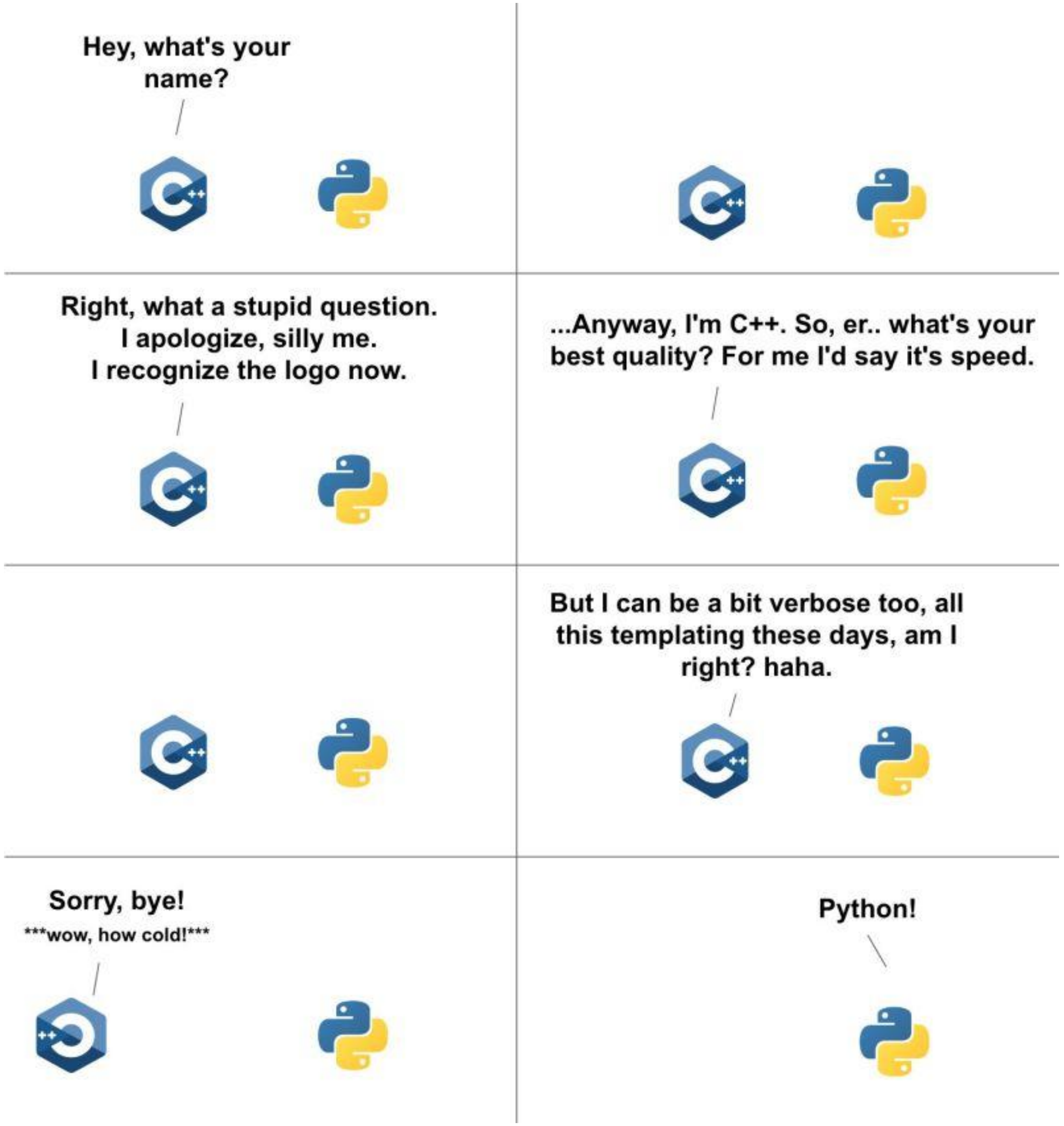
Python Performance

Joey Bernard – November 2022

Python is slow – isn’t it?

- A common complaint is that Python is slow
- Is this true?
- Like everything else, the answer is yes and no


Hey, what's your name?
Right, what a stupid question. I apologize, silly me. I recognize the logo now.
...Anyway, I'm C++. So, er.. what's your best quality? For me I'd say it's speed.
But I can be a bit verbose too, all this templating these days, am I right? haha.
Sorry, bye!
wow, how cold!
Python!

Using Python incorrectly

- In scientific computing, there is a huge amount of history from C/C++ and FORTRAN
- There are bad habits that are language specific
- Need to learn the idioms of the language you want to use
- Python is an untyped object-oriented language
- This means that Python always needs to inspect every object before a function can be applied
- What if we want to find the tangent of an array of values?

Straight Python

```
import time
import math

t0 = time.time()

size = 10000000
A = list(range(size))
B = list(range(size))
for i in range(size):
    B[i] = math.tan(A[i])

t1 = time.time()
print(t1 - t0)
```

Fixed Datatypes

```
import time
import math
import numpy

t0 = time.time()

size = 10000000
A = numpy.arange(size)
B = numpy.arange(size)
for i in range(size):
    B[i] = math.tan(A[i])

t1 = time.time()
print(t1 - t0)
```

Full numpy

```
import time
import math
import numpy

t0 = time.time()

size = 10000000
A = numpy.arange(size)

B = numpy.tan(A)

t1 = time.time()
print(t1 - t0)
```

Comparisons?

Code Version (10,000,000 elements)	Time (s)
Pure Python	2.361333
Numpy datatypes	3.466689
Numpy functions	0.152022

Jupyter timing

```
%%timeit
C = np.zeros((rows,cols))

for i in range(rows):
    for j in range(cols):
        for k in range(rows):
            C[i][j] += A[k][i] * B[j][k]
```

```
6.14 s ± 53 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Jupyter timing

```
%%timeit
C = A * B
```

```
20 µs ± 291 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

Other Interpreters?

- Python is a language specification
- Cpython is just the default implementation from the Python consortium
- There are other options – pypy is popular
- Issue is that not all modules are available, you may need to do some manual installation/tweaking

Code Version	Cpython time	Pypy time
Array.py	2.146039	0.763002
Numpy_array.py	0.158998	0.166613

What about parallel programming

- A common “solution” to speed issues is to parallelize your code
- This is a pair of undergraduate course in CS
- Two broad categories : multithreading and multiprocessing
- Multithreading – multiple threads within a single process
- Multiprocessing – multiple processes, talking to each other

Multithreading and the GIL

- In Cpython, we have the Global Interpreter Lock (GIL)
- This means that only one thread can run on the CPU at a time
- No speed up if it is CPU bound
- Great if code is IO bound
    - Can have threads start IO tasks and wait for them to finish
    - This allows another thread to run on the CPU instead of everybody waiting for IO
- There are GIL-less interpreters, but not universal yet

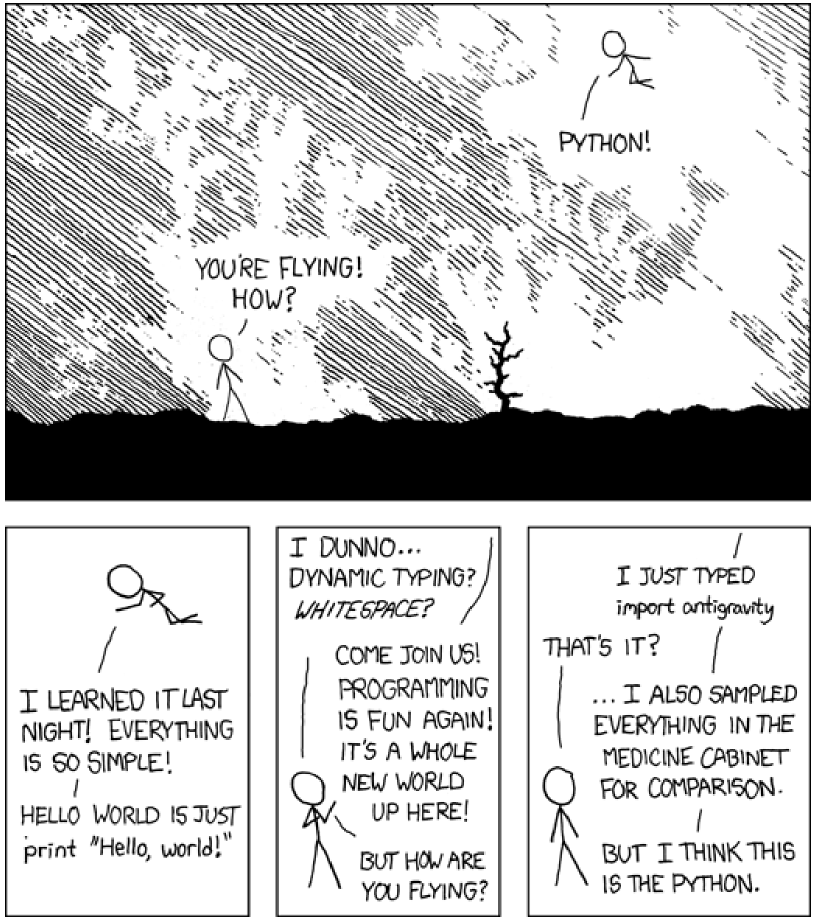
Multiprocessing – multiple GILs?

- We can get around the GIL by spreading the work across multiple processes
- Each process uses a separate instance of the interpreter
- This involves breaking your algorithm into discrete independent parts
- The trade off becomes number of processes vs amount of communication

```
from multiprocessing import Process


def f(name):
        ('hello', name)



if __name__ == '__main__’:
        p = Process(target=f, args=('bob',))
        p.start()
        p.join()
```

Modules?

- Before doing too much work, look
- The pYthon community of available packages is huge
- https://pypi.org currently has 414,076 packages


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Cython – for when you have to do it yourself

- Let’s say that your research is so cutting edge, nobody has ever done anything similar
- There is the option to write code in a lower level language for highly tuned algorithms
- The default binding is for C, but other options exist

Fibonacci example

```
from __future__ import print_function

def fib(n):
    """Print the Fibonacci series up to n."""
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a + b

    print()
```

How to compile

- First you need cython
  - Python –m pip install cython
- You also need a C compiler
- Then you need a setup.py file to manage the compilation
  - python setup.py build_ext –inplace
- This gives you a binary file that you can import, just like any other module

Setup.py

- from setuptools import setup
- from Cython.Build import cythonize

- setup(
- ext_modules=cythonize("fib.pyx"),
-)

Comparison

Parameter size	Python time	Cython time	Ratio
1000	618 ns	326 ns	1.8957
10000	853 ns	468 ns	1.8226
100000	1060 ns	596 ns	1.7785

Partial Compilations?

- What if you only have parts of your code that need to be optimized?
- You can use a Just-In-Time (JIT) compiler
- A popular one is numba
- You can use the JIT to compile individual functions that might need to be optimized

```
import time
import math
from numba import jit

@jit
def arr(size):
    A = list(range(size))
    B = list(range(size))
    for i in range(size):
        B[i] = math.tan(A[i])

t0 = time.time()
arr(10000000)
t1 = time.time()
print(t1 - t0)
```

Comparison

Array size	Python time	Pypy time	Numba time
100000	0.026575	0.012557	0.221454
1000000	0.261230	0.068846	0.264646
10000000	2.520349	0.750271	0.746917
100000000	died	42.667011	7.067443

Conclusions

What order should you try things for optimized code?

- Write correct Python
- Use optimized modules (Most people can stop here)
- Use a different interpreter or a JIT
- Hand-code lower level code to import