Performance Improvements with Cython

Kaivalya Rawal

Singapore Python User Group
October 2018 Meetup

What this talk is about

- A very brief introduction to Cython, from an amateur enthusiast's perspective
- Motivation and reasons to use Cython, based on my recent work experience with The Institute for Artificial Intelligence, University of Bremen.
- Tips on Cython optimisation and troubleshooting when working with large codebases

What this talk is not about

- Providing a thorough outline of Cython capabilities
- Describing best practices for Cython development
- Expert insights from experienced Cython developers

Live Demonstration 1

Fibonacci Sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 …

Generating the Nth fibonacci number: **O(n)**

Generating the first N fibonacci numbers: **O(n)** → **$O(n^2)$**

Live Demonstration 1

Fibonacci Sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 …

Generating the Nth fibonacci number: **O(n)**

Generating the first N fibonacci numbers: **O(n)** → **O(n^2)** *for "slow version"*

Demo: Speeding up **"Slow Fibonacci"** with Cython, to find the 34th Fibonacci number

Programming Languages

A human-computer **contract**, ie a set of instructions a **human** can write for a **computer** to perform.

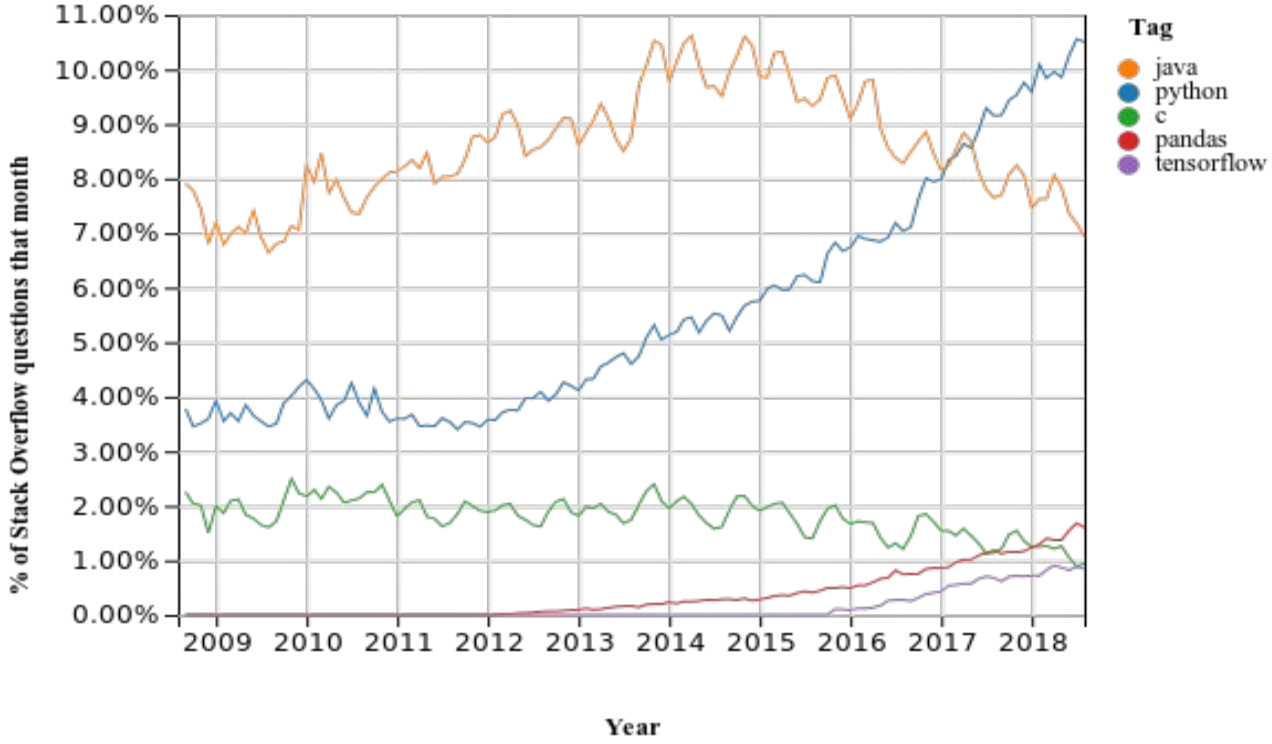
Programming Languages

A human-computer **contract**, ie a set of instructions a **human** can write for a **computer** to perform.

(Important?) Variations:

- Compiled vs Interpreted;
- Object Oriented vs Functional;
- *Static vs Dynamic;*
- *...*

Programming Languages



2018: Programming for Humans

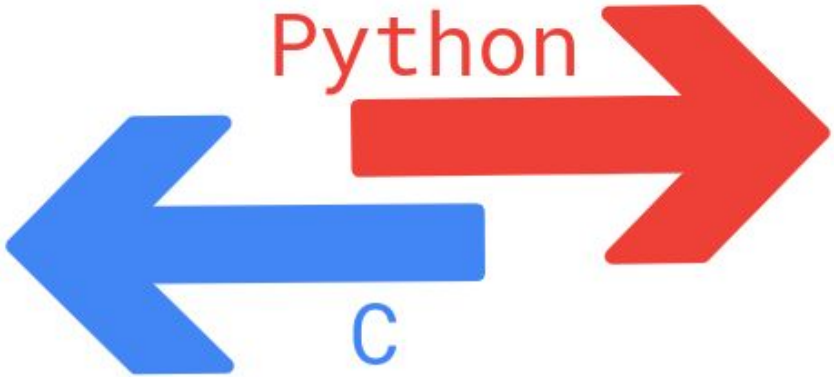
Manual Memory Allocation

Terse Syntax

Lower Level Control

High Speed

Contract is
“Computer Friendly”



Automatic Memory Allocation

Simple Syntax

Relatively Higher Level
Language Features

Lower Speeds

Contract is
“Human Friendly”

contracts are increasingly human-human too

2018: Programming for Humans

Manual Memory Allocation

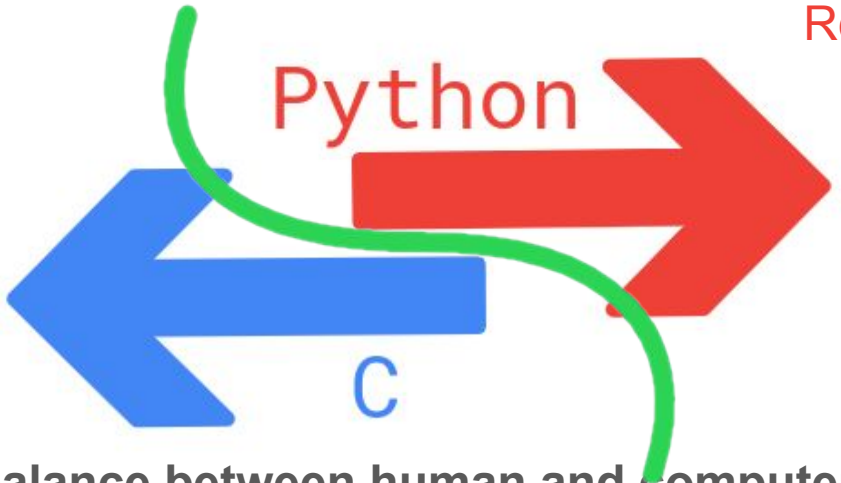
Terse Syntax

Lower Level Control

High Speed

Contract is
“Computer Friendly”

Cython



Automatic Memory Allocation

Simple Syntax

Relatively Higher Level
Language Features

Lower Speeds

Contract is
“Human Friendly”

strike a balance between human and computer friendly

Cython

A different implementation of the same contract (as CPython).

Cython ≠ CPython

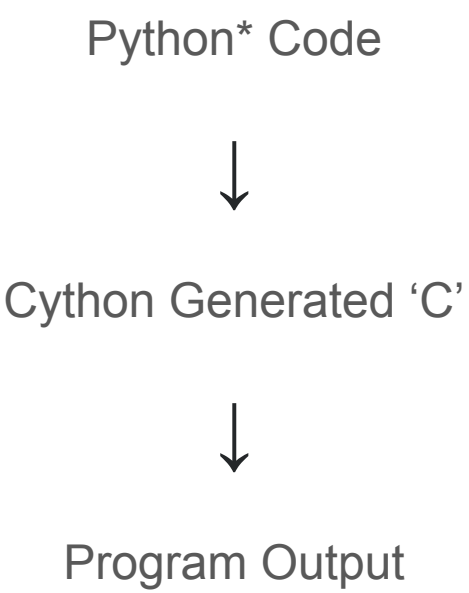
A *superset* of Python.

Cython

A different implementation of the same contract (as CPython).

Cython ≠ CPython

A *superset* of Python.

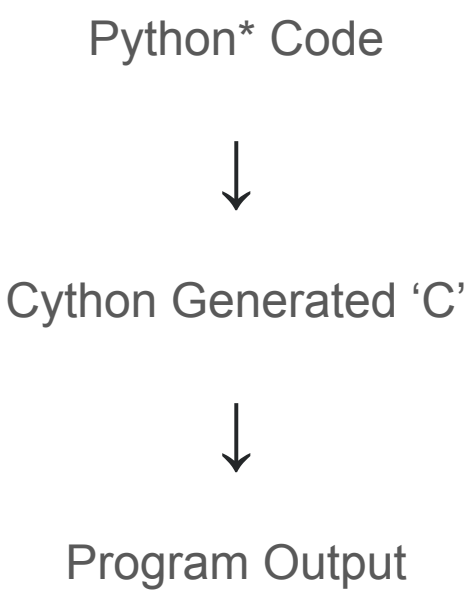


Live Demonstration 2

Fibonacci again

Extra steps:

1. Change extension to *.pyx*
2. Create *setup.py*
3. Compile with *build_ext --inplace*



Cython - Overview

Examples

1. n→int n
2. c→char c
3. d→dict d
4. def fib→cpdef fib
5. def fib→cdef fib
6. class C→cdef class C
    a. Create *.pxd* file
7. with NoGIL
8. Direct interfacing with C structs / procedures

Cython - Overview

Examples

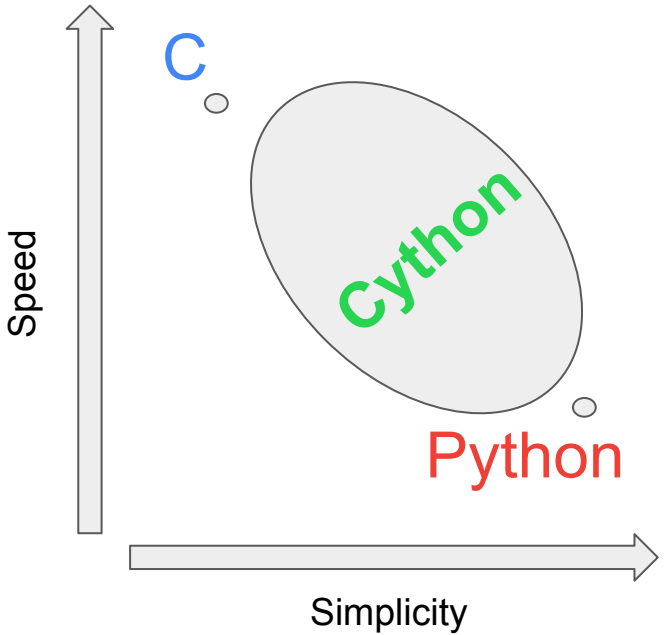
1. n→int n
2. c→char c
3. d→dict d
4. def fib→cpdef fib
5. def fib→cdef fib
6. class C→cdef class C
   a. Create *.pxd* file
7. with NoGIL
8. Direct interfacing with C structs / procedures

Concepts

1. Static typing (variable)
2. Static typing (variable)
3. Static typing (variable)
4. Static typing (params and return)
5. Static typing (C only)
6. Static typing (attributes)
7. ?
8. ?

Why use Cython?

- Customizable balance between Speed and Simplicity
- Incrementally speed up existing code, without the need for major redesigns*
- Generation of *potentially* higher quality C
- Faster development times



Cython - Optimising Large Codebases

Certainly possible: scikit-learn, scipy, pandas, pracmln, etc already use it in production

Still at version *0.30*, not a complete* superset yet! But still *Turing Complete*.

Cython - Optimising Large Codebases

Certainly possible: scikit-learn, scipy, pandas, pracmln, etc already use it in production

Still at version *0.30*, not a complete* superset yet! But still *Turing Complete*.

Two techniques to help “cythonize” existing code:

- Annotation
- Profiling

Live Demonstration 3

Annotations

- In a Jupyter Notebook
- In standalone files

Annotations vs Profiling

“Python Interaction” is not a reliable metric. (80-20 rule)

Lead developer might know better than the annotation tool which parts of the code deserve optimisation.

Annotations vs Profiling

“Python Interaction” is not a reliable metric. (80-20 rule)

~~Lead developer might know better than the annotation tool which parts of the code deserve optimisation.~~

“Premature optimization is the root of all evil” - Donald Knuth

“Never optimize without having profiled. Let me repeat this: Never optimize without having profiled your code. Your thoughts about which part of your code takes too much time are wrong.”

Live Demonstration 4

Profiling

Use built-in *cProfile* profiler

Extra steps:

1. Add compiler directive
2. External analyser - *cprofilev* / *snakeviz*

Getting Cython Help

Getting Cython Help

Getting Cython Help

Getting Cython Help

cython-users@googlegroups.com

kaivalyar.github.io/gsoc18-pracmln/

linkedin.com/in/kaivalyar/

Thank You

Questions?

kaivalyar.github.io/gsoc18-pracmln/

linkedin.com/in/kaivalyar/