SymEngine: Leveraging The Power Of A Computer Algebra System (CAS) To Another - Shikhar Jaiswal

About the speaker
Shikhar is currently a second year undergraduate at IIT Patna, majoring in Computer Science and Engineering. He is a contributor to SymPy, SymEngine as well as SymEngine.py. He has successfully completed a Google Summer of Code 2017 project for SymEngine titled 'Improving SymEngine's Python Wrappers and SymPy-SymEngine Integration', under SymPy, with Sumith Kulal (4th Year CSE Undergraduate at IIT Bombay) as one of the mentors.

Abstract
SymPy is a widely used symbolic manipulation library in Python. While it turns out to be very useful for many applications, one of the long-term problems with SymPy has been that the speed might be insufficient while handling very large expressions. Another problem is that due to being written in Python, it can be cumbersome to use from other languages like Julia, Ruby, JavaScript or C++, because it requires, say, a Python to Julia bridge, which might not always be robust and which inflicts additional overheads.

Methods
For these reasons, we implemented SymEngine, an Open-Source C++ symbolic manipulation library, with the goal of being the fastest library for symbolic manipulation, and allowing the use to other languages like Python (the most mature wrapper), Ruby, Julia, Haskell and C.

Results
We will show benchmarks comparing other popular Computer Algebra Systems (both open-source and commercial).

We will talk about why we chose C++ and what rules to follow so that the code cannot have an undefined behavior in Debug mode (thus providing similar ease of development as one is used to from Python), while being blazingly fast in Release mode.

The talk will focus heavily on SymEngine's use in SymPy, Sage and PyDy software (all three being Python based software). Detailed examples will be presented on various mathematical functionalities implemented, through SymPy and SymEngine created objects, as well as the details of our use of Cython language (not to be confused with CPython) in wrapping C++ data structures and creating Python and Cython classes as per the requirements. Some amount of time will also be devoted to the issues faced, and the solutions implemented while wrapping off the library in Python. We will also present a road-map portraying the present as well as previous strategies on porting SymPy on top of SymEngine.

Conclusion
SymEngine should fix the slowness of SymPy, while providing a familiar interface, and at the same allowing many languages to use it, thus creating a common platform/tool that many projects (like SymPy, Sage, PyDy ...) can use as their main symbolic engine and all contribute back to it.