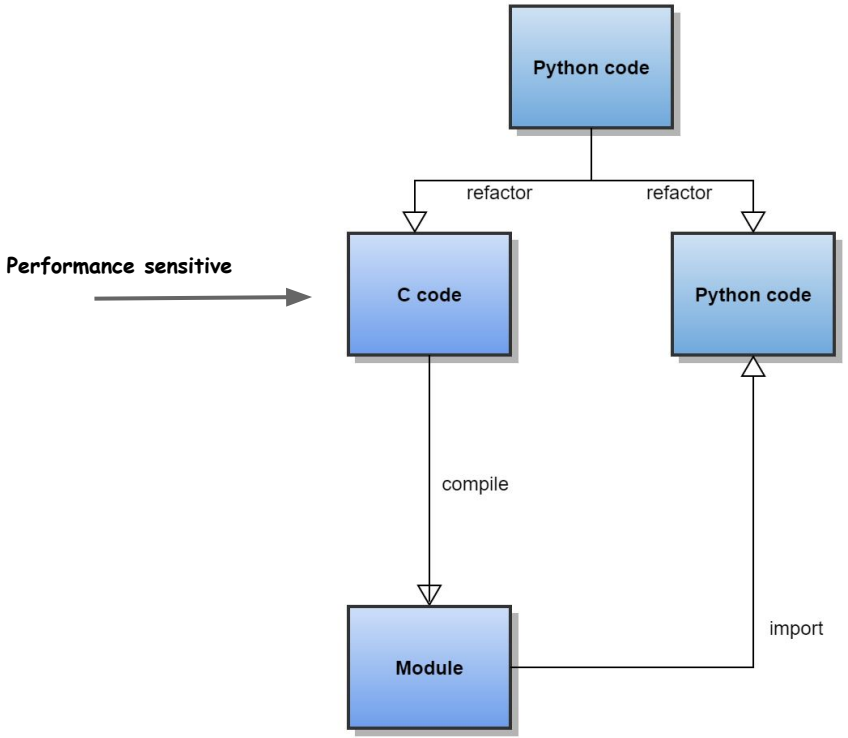
Accelerating Python

Hands-on with Cython

Methodology

1. Profile code
2. Optimize existing implementation
3. Explore different options:
    - Parallelizing the code (MPI, OpenMP, CUDA C, ...)
    - Using JIT compilers e.g. Numba, PyPy
    - Writing extensions (C/C++/Fortran/...)

Python Extension Modules



Python Extension Tools

- Python/C API
- C/C++ to python tools: ctypes, CFFI, PyBind11, SWIG, ...
- Fortran to python: f2py, f90wrap, ...
- Python to C/C++ intermediate languages: Pyrex, **Cython**

What is Cython?

- Cython is (almost) **python** plus **C data types**
- Enables conversion of python into C code
- Automates extension creation and compilation
- Provides superior performance to python

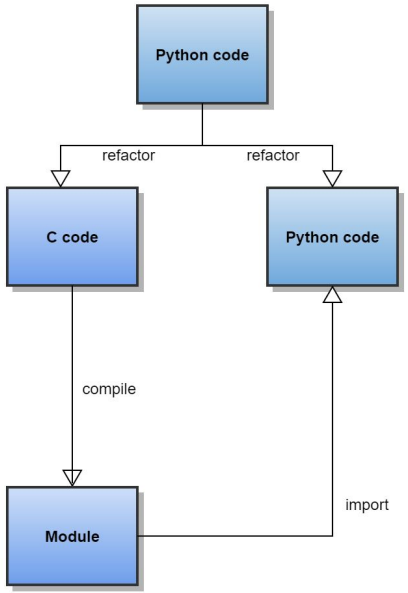
Exercise I: hello world

Requirements:

- **Cython**
- **C compiler**
- **Python-dev**
- **Numpy-dev**

Ex-I:

- **git clone https://github.com/MolSSI-Education/cython_intro.git**
- **cd cython_tutorial/Ex1**
- Execute: **cythonize -3 --inplace hello.pyx**
- Import hello module from the python interpreter

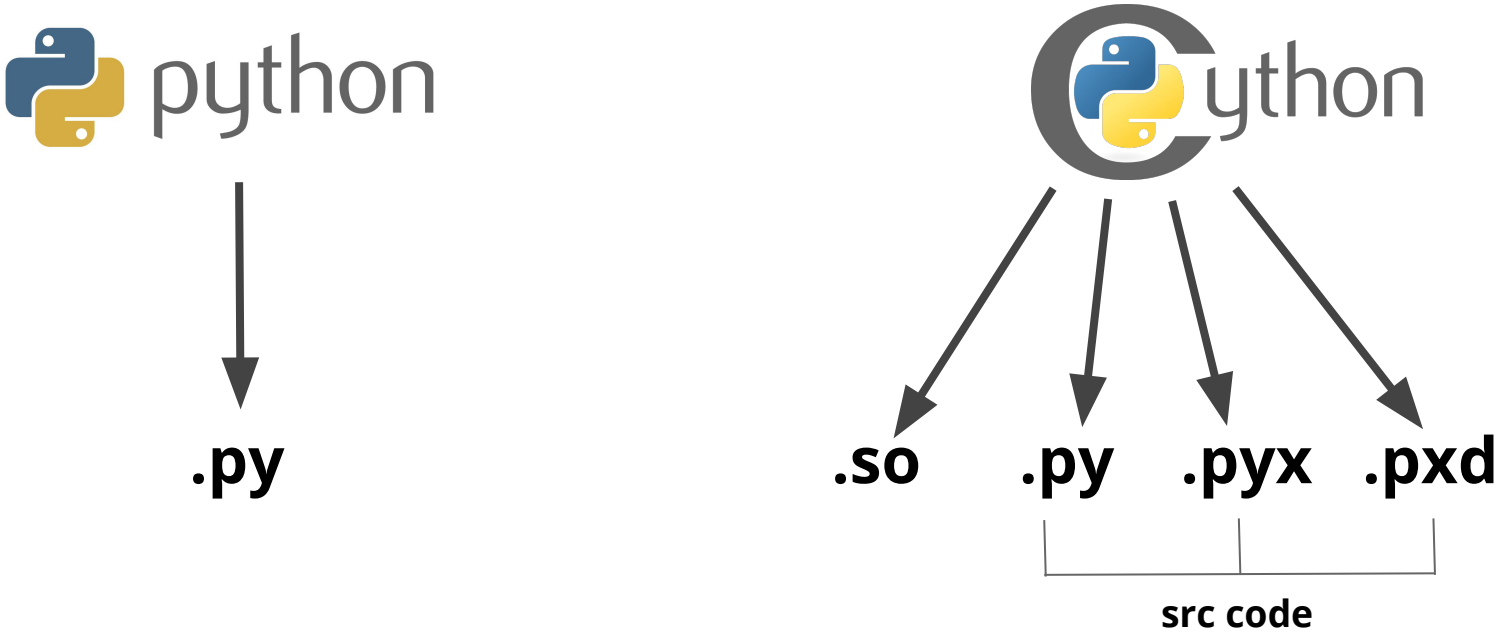


Exercise II: Cythonizing for loops

Ex-II:

- **cd ../Ex2**
- Create **cymod.pyx** from **pymod.py**
- Run **compare.py**
- Optimize **cymod.pyx**

Python vs Cython Pkg Files



Exercise III: Vectorization with NumPy

Ex-III:

- **cd ../Ex3**
- Run **python setup.py build --inplace**
- Vectorize the code in **pymod.py** in a new file **pymod_opt.py**
- Run **compare.py**

Questions?

Email me:

andrewabimansour@vt.edu

Andrew Abi-Mansour