Todd Leonhardt
todd.leonhardt@gmail.com

Interfacing C/C++ with Python - Lab

Hands-on with Cython and SWIG

Overview

- Lecture Slides
    - Learn to leverage strengths of C/C++ and Python
    - While alleviating the weaknesses
    - Efficiently integrate them to maximize productivity
    - Demonstrates basic usage of Cython, SWIG, and CFFI

- Exercises
    - Hands-on guided programming exercises using
        - SWIG (Simplified Wrapper and Interface Generator)
        - Cython (C-Extensions for Python)

Goals & Tools

- Gain familiarity with SWIG and Cython
  - Cover a few scenarios that would be useful in real world
  - Learn enough so you know how/where to learn more

- Tools Needed
  - Python 3.4 or newer (recommend Anaconda distro)
  - C and C++ compiler toolchains
  - SWIG and Cython (Cython comes with Anaconda)
  - Setup instructions available on GitHub
  - Excercise code is also on GitHub

PyPy Interlude

- Exercises don’t involve PyPy
  - But it can be good to know about it anyways
- PyPy
  - A fast, compliant alternative implementation of Python
  - With a Just-in-Time (JIT) compiler
  - Advantages
    - Big speedup compared to normal CPython
    - No code changes necessary
    - Works very well with CFFI
  - Disadvantages
    - Not compatible with all 3rd party Python libraries

Cython Exercises

- cython/integrate
  - Serves as a basic intro to using Cython for optimization
  - Details a typical series of steps for you to follow
  - Introduces Cython annotation HTML files

- cython/wrap_arrays
  - Serves as an intro to using Cython to wrap C code
  - Details a typical series of steps for you to follow
  - Deals with how to wrap functions which take pointers

SWIG Exercises

- swig/fastlz
  - Intro to wrapping existing C library with SWIG
  - Uses same fastlz C lib as cython/wrap_arrays exercise
  - Example of how to wrap STL vectors
  - Also example of how to link to dynamic libraries
- swig/logger
  - Intro to how to achieve cross-language polymorphism
  - Also covers how to wrap STL strings (std::string)
  - Universal Logger from Python and C++
    - Using Python logging module and SWIG directors

Instructions

- Available from GitHub:
  - Top-level instructions are here
  - Look at Readme.md markdown files for each exercise

- Solutions
  - On “solutions” branch
  - Don’t peak until you have really tried on your own

Cython for optimizing Python

Lab 1 - cython/integrate

https://github.com/tleonhardt/Python_Interface_Cpp/tree/master/cython/integrate

cython/integrate cyintegrate.pyx

```
from libc.math cimport cos
import cython


cdef double f(double x):
    return cos(x)

@cython.cdivision(True)
cpdef double integrate_f(double a, double b, int N):
    """Numerically integrate function f starting at point a and going to point b, using N rectangles.

    :param a: float - starting point
    :param b: float - ending point
    :param N: int - number of points to use in the rectangluar approximation to the integral
    :return: float - approximation to the true integral, which improves as N increases
    """
    cdef double s, dx
    cdef int i

    s = 0.0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

Cython for wrapping C/C++

Lab 2 - cython/wrap_arrays

https://github.com/tleonhardt/Python_Interface_Cpp/tree/master/cython/wrap_arrays

cython/wrap_arrays cyfastlz.pxd

```
from libc.stdint cimport uint8_t


cdef extern from "fastlz.h":
   int fastlz_compress(const uint8_t* inBuf, int length, uint8_t* output)
   int fastlz_decompress(const uint8_t* inBuf, int length, uint8_t* output, int maxout)
```

cython/wrap_arrays - compress

```
cpdef bytes compress(bytes in_buf):
    cdef int N, M

    N = len(in_buf)

    # The minimum input buffer size is 16.
    if N < 16:
        return None

    # The output buffer must be at least 5% larger than the input buffer and can't be smaller than 66 bytes
    M = max(int(1.5*N), 66)

    # Create the output buffer
    output = bytearray(M)

    # wrap byte arrays to c arrays for the call
    fcret = cfastlz.fastlz_compress(<const uint8_t*>in_buf, N, <uint8_t*> output)

    if fcret <=0:
        return None

    # Return the compressed data as a bytes object
    return bytes(output[:fcret])
```

cython/wrap_arrays - decompress

```
cpdef bytes decompress(bytes in_buf):
    cdef int N, M

    N = len(in_buf)

    # Bounds check length, just make sure it is positive
    if N < 1:
        return None

    # Create an output buffer of sufficient size
    M = max(int(4*N), 66)
    output = bytearray(M)

    # wrap byte arrays to c arrays for the call
    fcret = cfastlz.fastlz_decompress(<const uint8_t*> in_buf, N, <uint8_t*> output, M)

    # If error occurs, e.g. the compressed data is corrupted or the output buffer is not large enough, then 0 (zero)
    if fcret <=0:
        return None

    # Return the uncompressed data as a bytes object
    return bytes(output[:fcret])
```

SWIG for wrapping C/C++

Lab 3 - swig/fastlz

https://github.com/tleonhardt/Python_Interface_Cpp/tree/master/swig/fastlz

swig/fastlz test_swig.py

```
# Convert the text to a VectorUint8
text_vec = VectorUint8(text.encode())

# Create a vector to store the compressed data
compressed_vec = VectorUint8(len(text_vec) * 2)

# Compress the input text
success = Compress(text_vec, compressed_vec)

# Create a vector for the reconstructed text
recon_vec = VectorUint8(len(text_vec) * 2)

# Decmopress the compressed text to reconstruct a vector of original bytes
success = Decompress(compressed_vec, recon_vec)

# Convert the reconstructed text to a bytes
recon_bytes = bytes(recon_vec)

# And finally back to a str
reconstructed = recon_bytes.decode()
```

SWIG for bi-directional cross-language polymorphism (i.e. magic)

Lab 4 - swig/logger

https://github.com/tleonhardt/Python_Interface_Cpp/tree/master/swig/logger

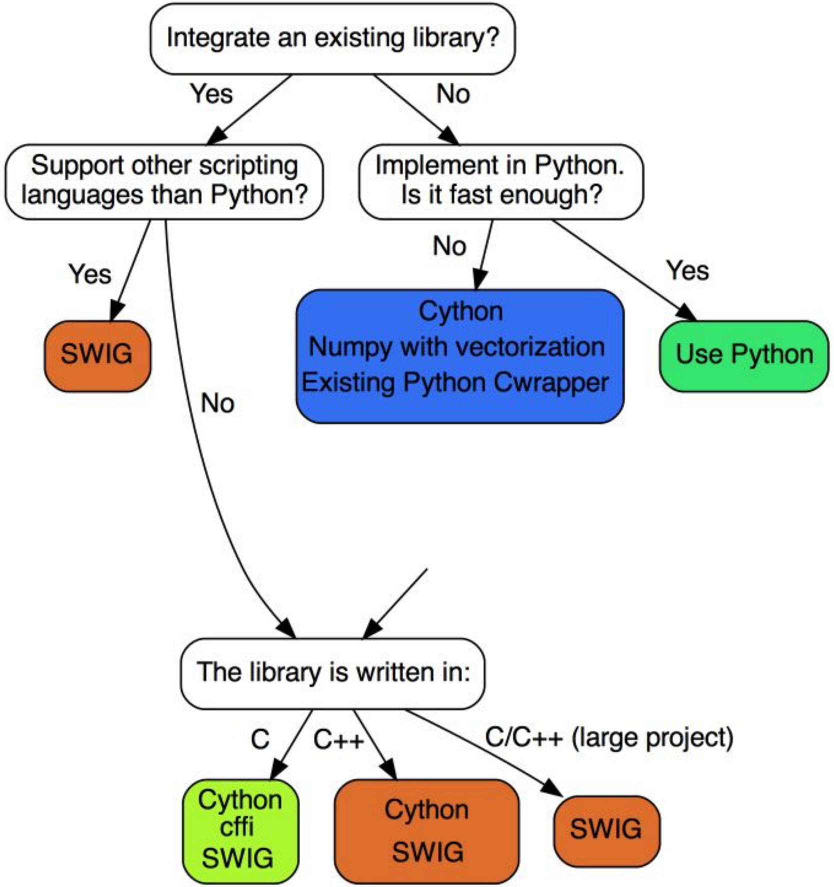
swig/logger runme.py

```
## Add a Python Logger (log owns the logger, so we disown it first by calling __disown__).
print()
print("Adding and calling a Python Logger")
print("----------------------------------")
log.setLogger(PyLogger().__disown__())
log.war("World")
log.delLogger()

# Let's do the same but use the weak reference this time.
print()
print("Adding and calling another Python logger")
print("------------------------------------------")
logger = PyLogger().__disown__()
log.setLogger(logger)
log.err("Cross language polymorphism in SWIG rocks!")
log.delLogger()
```

Final Thoughts

Which tool should I use?


Integrate an existing library?
Yes
No
Support other scripting languages than Python?
Implement in Python. Is it fast enough?
Yes
No
No
Yes
SWIG
Cython
Numpy with vectorization
Existing Python Cwrapper
Use Python
The library is written in:
C
C++
C/C++ (large project)
Cython
cffi
SWIG
Cython
SWIG
SWIG

- Like most things in engineering
    - There are tradeoffs …
        - So it depends

Where to learn more

- Cython
  - Main Site: http://cython.org
  - Documentation: http://docs.cython.org
  - 4 hour training video from SciPy 2015 with code
- SWIG
  - Main Site: http://www.swig.org
  - Documentation: http://www.swig.org/Doc3.0
  - 40 minute training video from Univ. Oslo with code
- PyPy
  - Main Site: http://pypy.org
  - Didn’t cover much, but can provide some easy wins