CythonBootstrap

March 31, 2015

1 Cython – A Transcompiler Language

1.1 Transform Your Python !!

1.1.1 By James Bonanno, Central Ohio Python Presentation, March 2015

There are many cases where you simply want to get speed up an existing Python design, and in particular code in Python to get things working, then optimize (yes, early optimization is the root of all evil, but it's even **more sinister** if you run out of ways to optimize your code.)

What is is good for?

- for making Python faster,
- for making Python faster in an easy way
- for wrapping external C and C++
- making Python accessible to C and C++ (going the other way)

This presentation seeks primarily to discuss ways to transform your Python code and use it in a Python project.

1.1.2 References

The new book by Kurt Smith is well written, clear in explanations, and the best overall treatment of Cython out there. An excellent book !! The book by Gorelick and Ozsvald is a good treatment, and it compares different methods of optimizing python including Shedskin, Theano, Numba, etc.

1] Kurt W. Smith **Cython, A Guide for Python Programmers**, O'Reilly, January 2015

2] Mich Gorelick & Ian Ozsvald **High Performance Python – Practical Performant Programming for Humans** O'Reilly September 2014

3] David Beazley and Brian K Jones, **Python Cookbook**, 3rd Edition, Printed May 2013, O'Reilly – Chapter 15, page 632

1.1.3 Why CYTHON?

It's more versatile than all the competition and has a manageable syntax. I hihgly recommend Kurt Smith's book on Cython. It's thorough, and if you read chapter 3, you will take in the essence of working with Cython functions. ***

Make sure to check out the new, improved documentation for Cython at:

http://docs.cython.org/index.html

This presentation will focus on using Cython to speed up Python functions, with some attention also given to arrays and numpy. There are more sophisticated treatments of using dynamically allocated memory, such as typically done with C and C++.

A good link on memory allocation, where the heap is used with malloc():

http://docs.cython.org/src/tutorial/memory_allocation.html?highlight=numpy

1.2 Getting Started:: Cython function types...

You must use "cdef" when defining a type inside of a function. For example,

```
def quad(int k):
    cdef int alpha = 1.5
    return alpha*(k**2)
```

People often get confused when using def, cdef, and cpdef.
The key factors are

- def is importable into python
- cdef is importable into C, but not python
- cpdef is importable into both

1.3 Getting Started:: *Cythonizing* a Python function

Now, if you were going to put pure cython code into action within your editor, say Wing IDE or PyCharm, you would want to define something like this in a file say for example ** cy_math.pyx **
Now, let's start with the familiar Fibonacci series ...

```
import cython

def cy_fib(int n):
    """Print the Fibonacci series up to n."""
    cdef int a = 0
    cdef int b = 1
    cdef int index = 0
    while b < n:
        print ("%d, %d, \n" % (index, b) )
        a, b   = b, a + b
        index += 1
```

1.4 Getting Started:: A Distutils setup.py ...

```
from distutils.core import setup, Extension
from Cython.Build import cythonize

#=======================================
# Setup the extensions
#=======================================
sources = [ "cyMath.pyx", "helloCython.pyx","cy_math.pyx", "bits.pyx", "printString.pyx"]

for fileName in sources:
    setup(ext_modules=cythonize(str(fileName)))

or...

map(lambda fileName : setup(ext_modules=cythonize(str(fileName))), sources)

In [1]: %%file ./src/helloCython.pyx

        import cython
        import sys

        def message():
```

```
            print(" Hello World ....\n")
            print(" Hello Central Ohio Python User Group ...\n")
            print(" The 614 > 650::True")
            print(" Another line ")
            print(" The Python version is %s" % sys.version)
            print(" The Cython version is %s" % cython.__version__)
            print(" I hope that you learn something useful . . . .")

        def main():
            message()
```

```
Overwriting ./src/helloCython.pyx
```

In [2]:
```
%%file ./src/cyMath.pyx

        import cython

        def cy_fib(int n):
            """Print the Fibonacci series up to n."""
            cdef int a = 0
            cdef int b = 1
            cdef int c = 0
            cdef int index = 0
            while b < n:
                print ("%d, %d, \n" % (index, b) )
                a, b   = b, a + b
                index += 1
```

```
Overwriting ./src/cyMath.pyx
```

In [3]:
```
%%file ./src/printString.pyx

        import cython

        def display(char *bytestring):
            """ Print out a bytestring byte by byte. """

            cdef char byte

            for byte in bytestring:
                    print(byte)
```

```
Overwriting ./src/printString.pyx
```

In [4]:
```
%%file ./src/bits.pyx

        import cython

        def cy_reflect(int reg, int bits):
            """ Reverse all the bits in a register.
                reg     = input register
                r       = output register
            """
            cdef int x
            cdef int y
```

```
            cdef int r
            x = 1 << (bits-1)
            y = 1
            r = 0
            while x:
                    if reg & x:
                        r |= y
                    x = x >> 1
                    y = y << 1
            return r


        def reflect(self,s, bits=8):
            """ Take a binary number (byte) and reflect the bits. """
            x = 1<<(bits-1)
            y = 1
            r = 0
            while x:
                    if s & x:
                            r |= y
                    x = x >> 1
                    y = y << 1
            return r
```

```
Overwriting ./src/bits.pyx
```

In [5]:
```
%%file ./src/setup.py

        from distutils.core import setup, Extension
        from Cython.Build import cythonize

        #========================================
        # Setup the extensions
        #========================================
        sources = [ "./src/cyMath.pyx", "./src/helloCython.pyx",
                   "./src/cy_math.pyx", "./src/bits.pyx",
                   "./src/printString.pyx"]

        #for fileName in sources:
        #    setup(ext_modules=cythonize(str(fileName)))

        map(lambda fileName : setup(ext_modules=cythonize(str(fileName))), sources)
```

```
Overwriting ./src/setup.py
```

In [6]:
```
!python ./src/setup.py build_ext --inplace
```

```
Compiling ./src/cyMath.pyx because it changed.
Cythonizing ./src/cyMath.pyx
running build_ext
building 'src.cyMath' extension
x86_64-linux-gnu-gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -f
x86_64-linux-gnu-gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-Bsymbolic-functions -Wl,-z,
Compiling ./src/helloCython.pyx because it changed.
Cythonizing ./src/helloCython.pyx
```

```
running build_ext
building 'src.helloCython' extension
x86_64-linux-gnu-gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -f
x86_64-linux-gnu-gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-Bsymbolic-functions -Wl,-z,r
running build_ext
Compiling ./src/bits.pyx because it changed.
Cythonizing ./src/bits.pyx
running build_ext
building 'src.bits' extension
x86_64-linux-gnu-gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -f
x86_64-linux-gnu-gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-Bsymbolic-functions -Wl,-z,r
Compiling ./src/printString.pyx because it changed.
Cythonizing ./src/printString.pyx
running build_ext
building 'src.printString' extension
x86_64-linux-gnu-gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -f
x86_64-linux-gnu-gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-Bsymbolic-functions -Wl,-z,r
```

```
In [7]: from src import helloCython
        helloCython.message()
```

```
Hello World ...

 Hello Central Ohio Python User Group ...

 The 614 > 650::True
 Another line
 The Python version is 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2]
 The Cython version is 0.20.1post0
 I hope that you learn something useful . . . .
```

```
In [8]: from src import cyMath
        cyMath.cy_fib(100)
```

```
0, 1,

1, 1,

2, 2,

3, 3,

4, 5,

5, 8,

6, 13,

7, 21,

8, 34,

9, 55,

10, 89,
```

```
In [9]: from src import bits
        from bits import cy_reflect
        hexlist = [int(0x01),int(0x02),int(0x04),int(0x08)]
        [hex(cy_reflect(item,8)) for item in hexlist]


        ---------------------------------------------------------------------------

        ImportError                               Traceback (most recent call last)

        <ipython-input-9-1d683ed3449d> in <module>()
          1 from src import bits
    ----> 2 from bits import cy_reflect
          3 hexlist = [int(0x01),int(0x02),int(0x04),int(0x08)]
          4 [hex(cy_reflect(item,8)) for item in hexlist]


        ImportError: No module named bits


In [ ]: from src import printString
        printString.display('123')

In [ ]: # A little list comprehension here ...
        # A comparative method to the Cython printString function

        numberList = [1,2,3]
        [ord(str(value)) for value in numberList]
```

1.4.1 Now let's see the time difference between a cyfib and pyfib ...

```
In [ ]: %%file ./src/cyFib.pyx
        def cyfib(int n):
            cdef int a = 0
            cdef int b = 1
            cdef int index = 0
            while b < n:
                a, b = b, a+b
                index += 1
            return b
```

2 Introducing runcython !!

- Is located on Github
- Easy installation == pip install runcython
- Russell91 on Github

https://github.com/Russell91/runcython
There is a runcython and makecython function calls . . . . .

```
In [ ]: !makecython ./src/cyFib.pyx
```

```
In [ ]: def pyfib(n):
            a = 0
            b = 1
            index = 0
            while b < n:
                a, b = b, a+b
                index += 1
            return b
```

```
In [ ]: %timeit pyfib(1000)
```

```
In [ ]: import cyFib
        %timeit cyFib.cyfib(1000)
```

2.0.2 NOW THAT IS A CONSIDERABLE SPEEDUP ...

Fibonnaci function shows a factor of over **1500 %** Improvement
Let's take a look at disassembly for some reasons for this ....

```
In [ ]: import dis
        dis.dis(pyfib)
```

```
In [ ]: import cProfile
        cProfile.run('pyfib(1000)')
```

2.0.3 Another Example, with a polynomial this time ...

For now, lets begin with a polynomial function, and compare how to do this in python and cython! ....
Now consider a function such as
$f(x) = a_0x^n + a_1x^{(n-1)} + a_2x^{(n-2)}.....a_nx^0$
where in the case below n is selected as 2, and - $a_0 = 0.1$, - $a_1 = 0.5$ - $a_2 = 0.25$.
The cython function to do this called "cypoly" while the python version is called "pypoly". Each function is defined with a functional programming techinque of lambda and map, as shown below.

```
In [ ]: %%file ./src/cyPoly.pyx
        def cypoly(int n, int k):
            map(lambda x:(1.0*x**2 + 0.5*x + 0.25*x), range(k))
```

```
In [ ]: !makecython ./src/cyPoly.pyx
```

```
In [ ]: def pypoly(n,k):
            map(lambda x:.1*x**2 + .5*x + 0.25*x, range(k))
```

Now to compare the two ....

```
In [ ]: from src import cyPoly
        %timeit cyPoly.cypoly(4,5000)
        %timeit pypoly(4,5000)
```

2.0.4 Now's lets do something graphically, like plot a trig function. Let's also use a float/double type.

In [ ]:
```
%%file ./src/sineWave.pyx
import cython
from libc.math cimport sin

def sinewave(double x):
    """ Calculate a sinewave for specified number of cycles, Ncycles, at a given frequency."""
    return sin(x)
```

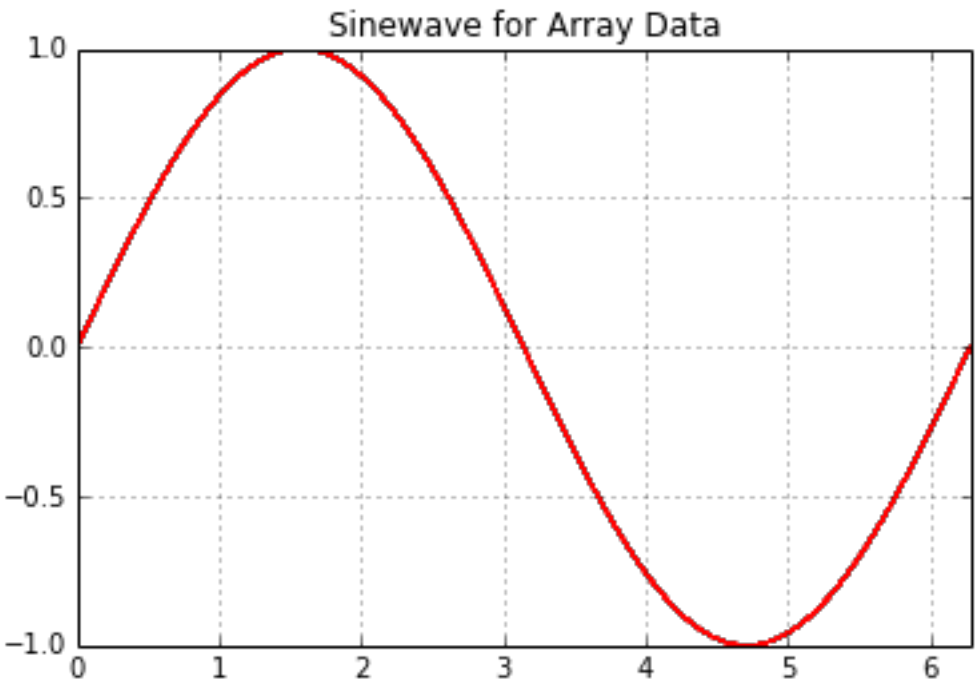
In [ ]:
```
!makecython ./src/sineWave.pyx
```

In [ ]:
```
from src import sineWave
import math
angle90 = math.pi/2
sineWave.sinewave(angle90)
```

2.0.5 Now let's looking a data that involves arrays, and look at both python and numpy versions as well.

In [15]:
```
%matplotlib inline

import numpy as np

x = np.linspace(0,2*np.pi,2000)
%timeit plot(x,np.sin(x),'r')

## %timeit plot(x,sineWave.sinewave(x),'r') <== Why is this a problem ??

xlim(0,6.28)
title('Sinewave for Array Data')
grid(True)
```

```
The slowest run took 36.06 times longer than the fastest. This could mean that an intermediate result is
1000 loops, best of 3: 744 µs per loop
```

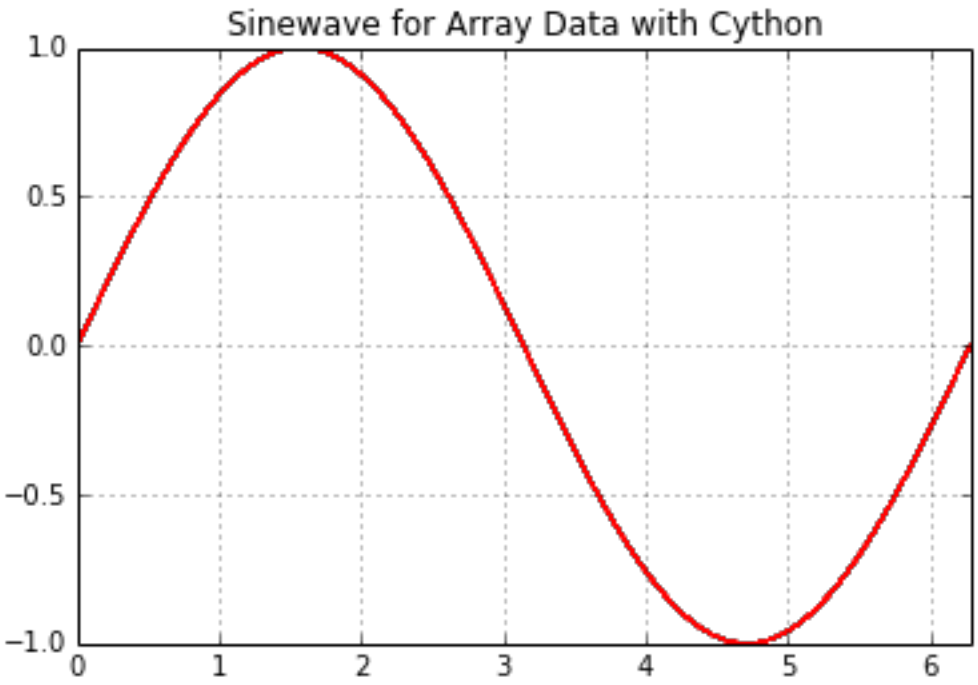


In [ ]:
```
%%file ./src/myFunc.pyx

import cython
import numpy as np
cimport numpy as np

@cython.boundscheck(False)
@cython.wraparound(False)
def myfunc(np.ndarray[double, ndim=1] A):
    return np.sin(A)
```

In [ ]:
```
!makecython ./src/myFunc.pyx
```

In [13]:
```
%matplotlib inline


from src import myFunc
import cython
import numpy as np

x = np.linspace(0,2*np.pi,2000)
y = myFunc.myfunc(x)

%timeit plot(x,y,'r')

xlim(0,6.28)
title('Sinewave for Array Data with Cython')
grid(True)
```

```
The slowest run took 45.37 times longer than the fastest. This could mean that an intermediate result is
1000 loops, best of 3: 686 µs per loop
```



3 Summary & Conclusions

This talk has presented the basics of getting started with Cython and IPython/Jupyter Notebook. There were examples presented on how to compile Cython programs with a setup.py and distuils as well as a nice application, runcython. Basic programs and some programs with arrays were demonstrated.

Cython is flexible, and it's flexibility is matched by it's performance.

It's realitively easy to use, but it does have some details to watch out for when working with arrays, references, etc.

Overall

- Cython enables Python code to be transformed easily
- The transformed Python code is signficantly faster
- Wide support and documentation exists for Cython
- Language has evolved and grown over the past few years with widespread support
- Usage in Ipython Notebook / Jupyter is now well supported
- Can be used on a wide variety of programs, ranging from math to logic.

Transform your Python with Cython !!

```
In [ ]: !python-config --cflags
```

```
In [ ]: !python-config --ldflags
```

```
In [ ]:
```