Cython

Cython is a programming language that aims to be a superset of the Python programming language, designed to give C-like performance with code that is written mostly in Python.

Cython works by producing a standard Python module. However, the behaviour differs from standard Python in that the module code, originally written in Python, is translated into C. And the resulting code is fast.

This is due the fact that Python is a dynamic programming language and moves many things at runtime whereas the C language takes care at compilation time only.

Installing Cython:

Unlike Python, Cython requires a C compiler int the system to run the code. For getting C compiler in various systems steps are:

1. Linux: The GNU C Compiler (gcc) is usually present, or easily available through the package system. On Ubuntu or Debian, for instance, the command **sudo apt-get install build-essential** will fetch everything you need.
2. Mac OS: To retrieve gcc, one option is to install Apple's XCode, which can be retrieved from the Mac OS X's install DVDs or from https://developer.apple.com/.
3. Windows: A popular option is to use the open source MinGW (a Windows distribution of gcc). Or a more popular option is to use Microsoft's Visual C.

The Simplest way of installing Cython is by tying the command on terminal for Linux or Mac & command prompt in Windows:

pip install Cython

For Running your 1st code through Cython:

Create a file named helloworld.pyx and write the following code into it:

```
Print ('Hello World')
```

After this create a new file name setup.py having the following code:

```
from setuptools import setup

from Cytohn.Build import cythonize


setup (
        ext_modules = cythonize('helloworld.pyx')
)
```

To use this to build your Cython file use the command line options:

Python setup.py build_ext –-inplace

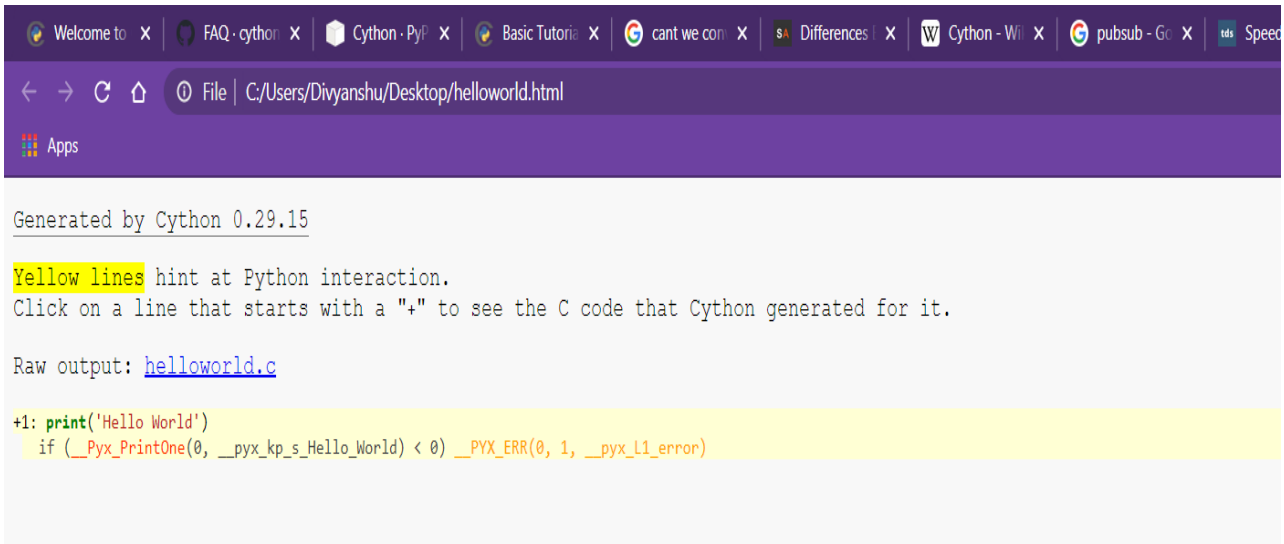
It will create two new files in our machine name helloworld.c and helloworld.

Cython has a way to visualise where interaction with Python objects and Python's C-API is taking place. For this, pass the annotate=True parameter to cythonize (). It produces a HTML file. For this make changes in setup.py file:

```
from setuptools import setup
from Cytohn.Build import cythonize

setup (
        ext_modules = cythonize ('helloworld.pyx',
                                 annotate=True)
)
```

This will produce one extra file name 'helloworld.html'



Basic Syntax:

Declaring basic Data Types:

```
cdef int i, j, k;
cdef float f, g [49], *h;
For declaring struct, union or enum:
Cdef struct Grail:
        int age
        float volume
cdef union Food:
        char *spam
        float *egg
cdef enum CheeseType:
        cheddar, edam, camebert
cdef enum CheeseState:
        hard=1
        soft=2
        runny=3
```

When we want to refer to them, we can simple write:

```
Cdef Grail *gp;
```

For defining functions:

```
Cdef int eggs (unsigned login l, float f):
        …
```

For defining class:

```
Cdef class Shruberry:
        Cdef int width, height
        Def __init__ (self, w, h):
                Self.width = w
                Self.height = h
```

Prime Number program in Python:

```
def primes_python(nb_primes):
    p = []
    n = 2
    while len(p) < nb_primes:
        # Is n prime?
        for i in p:
            if n % i == 0:
                break

        # If no break occurred in the loop
        else:
            p.append(n)
        n += 1
    return p
```

Prime number program In Cython:

```
def primes(int nb_primes):
    cdef int n, i, len_p
    cdef int p [1000]
    if nb_primes > 1000:
        nb_primes = 1000

    len_p = 0 # The current number of elements in p.
    n = 2
    while len_p < nb_primes:
        # Is n prime?
        for i in p [: len_p]:
            if n % i == 0:
                break
```

```
        # If no break occurred in the loop, we have a prime.
        else:
            p[len_p] = n
            len_p += 1
        n += 1

    # Let's return the result in a python list:
    result_as_list = [prime for prime in p [: len_p]]
    return result_as_list
```

Rusult of running these two:

python -m timeit -s 'from example_py import primes_python' 'primes_python (1000)'

10 loops, best of 3: 23 msec per loop

python -m timeit -s 'from example import primes' 'primes (1000)'

1000 loops, best of 3: 1.65 msec per loop

Advantages of using Cyton:

1. Working with external C libraries can be faster

Python packages like NumPy wrap C libraries in Python interfaces to make them easy to work with. However, going back and forth between Python and C through those wrappers can slow things down. Cython lets you talk to the underlying libraries directly, without Python in the way. (C++ libraries are also supported.)

2. You can use both C and Python memory management

If you use Python objects, they're memory-managed and garbage-collected the same as in regular Python. But if you want to create and manage your own C-level structures, and use malloc/free to work with them, you can do so. Just remember to clean up after yourself.

3. Cython can use Python type hinting syntax

Python has a type-hinting syntax that is used mainly by linters and code checkers, rather than the CPython interpreter. Cython has its own custom syntax for code decorations, but with recent revisions of Cython you can use Python type-hinting syntax to provide basic type hints to Cython as well.

4. Cython can be used to obscure sensitive Python code

Python modules are trivially easy to decompile and inspect, but compiled binaries are not. When distributing a Python application to end users, if you want to protect some of its modules from casual snooping, you can do so by compiling them with Cython. Note, though, this is a side effect of Cython's capabilities, not one of its intended functions.

Limitation of Cython

1. Little speedup for conventional Python code

When Cython encounters Python code it can’t translate completely into C, it transforms that code into a series of C calls to Python’s internals. This amounts to taking Python’s interpreter out of the execution loop, which gives code a modest 15 to 20 percent speedup by default. Note that this is a best-case scenario; in some situations, you might see no performance improvement, or even a performance degradation.