



Making your C library callable from Python by wrapping it with Cython

📅 Updated: May 03, 2018

Cython is known for its ability to increase the performance of Python code. Another useful feature of Cython is making existing C functions callable from within (seemingly) pure Python modules.

I have recently faced at work the need to directly interact from Python with a small C library I have written. After a little research, I decided that Cython was the most fitting choice out of several existing options. While it is rather simple (which is kind of the point) to wrap C library with Cython, I have found no direct and simple tutorial for this task, so I decided to share the process.

In this tutorial we will wrap a simple C function with Cython and call it from a pure Python module, which will be agnostic to the fact its calling C code. This will require the following steps:

1. Building the C library
2. Installing Cython
3. Creating a `.pyx` file in which the C function will be declared and wrapped
4. Creating a `setup.py` file which will create a shared object that will function as an importable python module
5. Building the module
6. Creating a pure Python file, importing the module, and calling the wrapped function

The code can be found here:

 **stavshamir / cython-c-wrapper**
(<https://github.com/stavshamir/cython-c-wrapper>)
Simple example of wrapping a C library with Cython
★ 20 (<https://github.com/stavshamir/cython-c-wrapper/stargazers>)
👤 10 (<https://github.com/stavshamir/cython-c-wrapper/network/members>)

1. Building the C Library

We will wrap the following function (files are here if you want to follow along). It is simple and not so useful, but you can literally wrap any function—which can be used to wrap lower-level system calls which are not exposed in Python!

```
void hello(const char *name) {  
    printf("hello %s\n", name);  
}
```

First, we need to build a library (it may be dynamic or static, for simplicity we will build it as a static library). Create an object file and build the library:

```
$ gcc -c examples.c  
$ ar rcs libexamples.a examples.o
```

Creating a makefile is always helpful for your future-self and coworkers, and will ease the build process of the Cython module:

```
CC = gcc  
  
default: libexamples.a  
  
libexamples.a: examples.o  
    ar rcs $@ $^  
  
examples.o: examples.c examples.h  
    $(CC) -c $<  
  
clean:  
    rm *.o *.a
```

2. Installing Cython

Simply install with pip:

```
$ pip3 install cython
```

3. Creating a pyx file

The pyx file is what Cython will compile to a shared object. This is the file which is written in the actual Cython language, a superset of Python, and as such mixes pure Python with C-like declarations. In the following snippet we include the declaration of the hello function, and wrap it with a Python-callable function:

```
cdef extern from "examples.h":
    void hello(const char *name)

def py_hello(name: bytes) -> None:
    hello(name)
```

Note: I am using type hints in the wrapper function (the `: bytes` and the `-> None`). They are not part of the Cython syntax and are not required. I tend to use them in the signature of (almost) every function I write, and this is why.

4. Creating a setup.py

Cython integrates with distutils, facilitating the build of the shared object:

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

examples_extension = Extension(
    name="pyexamples",
    sources=["pyexamples.pyx"],
    libraries=["examples"],
    library_dirs=["lib"],
    include_dirs=["lib"]
)
setup(
    name="pyexamples",
    ext_modules=cythonize([examples_extension])
)
```

Notice the *libraries*, *library_dir* and *include_dir* parameters—the library name is the file name without the lib prefix and .a suffix, and the paths to the directories should obviously match the project structure. However, if all files are in the same directory, the dir parameters may be omitted.

5. Build the module

From the CLI, run the following command:

```
$ python setup.py build_ext --inplace
```

The shared object will be compiled, and if everything went well, a 'build' directory, a 'pyexamples.c' file and a shared object with a somewhat complex name will be created. You only need the shared object, so feel free to remove the C file and the build directory. If you want, it is interesting to take a peek at the generated C file!

Again, writing a makefile is recommended. We can use the makefile we made earlier to build everything with one simple command:

```
LIB_DIR = lib

default: pyexamples

pyexamples: setup.py pyexamples.pyx $(LIB_DIR)/libexamples.a
    python3 setup.py build_ext --inplace && rm -f pyexamples.c && rm -Rf build

$(LIB_DIR)/libexamples.a:
    make -C $(LIB_DIR) libexamples.a

clean:
    rm *.so
```

6. Calling the wrapped function

Almost done! Now we can use our C function from Python:

```
import pyexamples

if __name__ == '__main__':
    pyexamples.py_hello(b"world")
```

As you see, we import the pyexamples module as if it was a regular Python module, and call the function as if it was a regular Python function—but do notice that we pass a bytes string and not a regular string, as C code essentially handles bytes strings (try using a unicode string and see what happens). Now run the module, and see the final result for your own.

I recommend deleting all the build-generated files, and trying to run the pure Python module, and then using the makefile to see how it eases the build process.

That's it!

You can now replace the simple hello function with more interesting functions and profit from the benefits of being able to directly call C from Python.

Some more information about wrapping C code with Cython can be found in the [official docs](http://cython.readthedocs.io/en/latest/src/userguide/external_C_code.html) (http://cython.readthedocs.io/en/latest/src/userguide/external_C_code.html).

LEAVE A COMMENT

0 Comments

stavshamir

 Login ▾

 Recommend

 Tweet

 Share

Sort by Best ▾

Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name

Be the first to comment.

 Subscribe  Add Disqus to your siteAdd DisqusAdd