무료 전자 책

배우기

cython

Free unaffiliated eBook created from
Stack Overflow contributors.

#cython

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: cython

It is an unofficial and free cython ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official cython.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: Cython

Cython ?

Cython , C C Python . Python C .

?

Cython cython - C C ++ . C . .

Python Cython CPython API . , . Cython C . Cython .

?

Cython Cython . Python .

Examples

Cython

`cython` (, NumPy) C - . `cython` C C .

1 : Cython

Cython . .

1. pip easy_install PyPI :

```
$ pip install cython
$ easy_install cython
```

2. :

```
$ conda install cython
```

3. enpkg Enthought :

```
$ enpkg cython
```

github .

```
$ python setup.py install
```

,

`cython` `cython3` . .

```
$ apt-get install cython cython3
```

Windows

Windows pip .whl . Windows .whl .

---

2 : C

Cython C C C ++ . gcc .

,

`build-essential` . .

```
$ sudo apt-get install build-essential
```

XCode gcc .

Windows

MinGW (Windows Minimalist GNU) gcc Windows . Visual Studio .

Cython pyx C (*cythonized*) . .

.

- `hello.pyx` Cython .
- `test.py` hello Python .
- `setup.py` Cython .

hello.pyx

```
from libc.math cimport pow

cdef double square_and_add (double x):
    """Compute x^2 + x as double.

    This is a cdef function that can be called from within
    a Cython program, but not from Python.
```

```
    """
    return pow(x, 2.0) + x

cpdef print_result (double x):
    """This is a cpdef function that can be called from Python."""
    print("({} ^ 2) + {} = {}".format(x, x, square_and_add(x)))
```

test.py

```
# Import the extension module hello.
import hello

# Call the print_result method
hello.print_result(23.0)
```

setup.py

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

# define an extension that will be cythonized and compiled
ext = Extension(name="hello", sources=["hello.pyx"])
setup(ext_modules=cythonize(ext))
```

`cython hello.pyx` C `gcc` . distutils :

```
$ ls
hello.pyx  setup.py  test.py
$ python setup.py build_ext --inplace
$ ls
build  hello.c  hello.cpython-34m.so  hello.pyx  setup.py  test.py
```

(.so) Python `test.py` .

```
$ python test.py
(23.0 ^ 2) + 23.0 = 552.0
```

Cython : https://riptutorial.com/ko/cython/topic/2925/cython-

2: C ++

Examples

DLL : C ++ Cython to Python

Cython C ++ dll . .

- Visual Studio C ++ DLL .
- Cython DLL Python .

Cython Python .

DLL Visual Studio DLL .

.

1. `complexFunLib.h` : C ++ DLL
2. `complexFunLib.cpp` : C ++ DLL CPP
3. `ccomplexFunLib.pxd` : Cython "header"
4. `complexFunLib.pyx` : Cython ""
5. `setup.py` : Cython `complexFunLib.pyd`
6. `run.py` : Cython wrapped DLL Python

C ++ DLL : `complexFunLib.h` `complexFunLib.cpp`

DLL . Visual Studio DLL C ++ . . `k ee k*exp(ee) . res` . . OpenMP OpenMP Visual Studio .

H

```
// Avoids C++ name mangling with extern "C"
#define EXTERN_DLL_EXPORT extern "C" __declspec(dllexport)
#include <complex>
#include <stdlib.h>

// Handles 64 bit complex numbers, i.e. two 32 bit (4 byte) floating point numbers
EXTERN_DLL_EXPORT void mp_mlt_exp_c4(std::complex<float>* k,
                                     std::complex<float>* ee,
                                     int sz,
                                     std::complex<float>* res,
                                     int threads);

// Handles 128 bit complex numbers, i.e. two 64 bit (8 byte) floating point numbers
EXTERN_DLL_EXPORT void mp_mlt_exp_c8(std::complex<double>* k,
std::complex<double>* ee,
                                     int sz,
                                     std::complex<double>* res,
                                     int threads);
```

CPP

```
#include "stdafx.h"
#include <stdio.h>
#include <omp.h>
#include "complexFunLib.h"

void mp_mlt_exp_c4(std::complex<float>* k,
                   std::complex<float>* ee,
                   int sz,
                   std::complex<float>* res,
                   int threads)
{
    // Use Open MP parallel directive for multiprocessing
    #pragma omp parallel num_threads(threads)
    {
        #pragma omp for
        for (int i = 0; i < sz; i++) res[i] = k[i] * exp(ee[i]);
    }
}

void mp_mlt_exp_c8(std::complex<double>* k,
                   std::complex<double>* ee,
                   int sz, std::complex<double>* res,
                   int threads)
{
    // Use Open MP parallel directive for multiprocessing
    #pragma omp parallel num_threads(threads)
    {
        #pragma omp for
        for (int i = 0; i < sz; i++) res[i] = k[i] * exp(ee[i]);
    }
}
```

Cython : **ccomplexFunLib.pxd complexFunLib.pyx**

C ++ DLL Cython . .

- Cython .
- DLL .

Python `.pyd` DLL Cython C ++ .

PXD

C ++ . , Cython . Cython . `ccomplexFunLib.pxd` `c` . .

```
cdef extern from "complexFunLib.h":
    void mp_mlt_exp_c4(float complex* k, float complex* ee, int sz,
                       float complex* res, int threads);
    void mp_mlt_exp_c8(double complex* k, double complex* ee, int sz,
                       double complex* res, int threads);
```

PYX

C ++ `cpp` . Numpy `ndarray` DLL . Cython `memoryview` , `ndarray` ( ).

```
cimport ccomplexFunLib  # Import the pxd "header"
# Note for Numpy imports, the C import most come AFTER the Python import
import numpy as np  # Import the Python Numpy
cimport numpy as np  # Import the C Numpy

# Import some functionality from Python and the C stdlib
from cpython.pycapsule cimport *

# Python wrapper functions.
# Note that types can be delcared in the signature

def mp_exp_c4(np.ndarray[np.complex64_t, ndim=1] k,
              np.ndarray[np.complex64_t, ndim=1] ee,
              int sz,
              np.ndarray[np.complex64_t, ndim=1] res,
              int threads):
    '''
    TODO: Python docstring
    '''
    # Call the imported DLL functions on the parameters.
    # Notice that we are passing a pointer to the first element in each array
    ccomplexFunLib.mp_mlt_exp_c4(&k[0], &ee[0], sz, &res[0], threads)

def mp_exp_c8(np.ndarray[np.complex128_t, ndim=1] k,
              np.ndarray[np.complex128_t, ndim=1] ee,
              int sz,
              np.ndarray[np.complex128_t, ndim=1] res,
              int threads):
    '''
    TODO: Python docstring
    '''
    ccomplexFunLib.mp_mlt_exp_c8(&k[0], &ee[0], sz, &res[0], threads)
```

: **`setup.py` `run.py`**

setup.py

Cython Python . `.pyd` . `setup.py` (: `complexFunLib.h` , `complexFunLib.dll` , `ccomplexFunLib.pxd` `complexFunLib.pyx`) .

```
. build_ext --inplace
```

`.pyd` . `.pyd` . `.pyd` `setup.py` .

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
import numpy as np

ext_modules = [
    Extension('complexFunLib',
              ['complexFunLib.pyx'],
              # Note here that the C++ language was specified
              # The default language is C
              language="c++",
```

```
                libraries=['complexFunLib'],
                library_dirs=['.'])
    ]

setup(
    name = 'complexFunLib',
    cmdclass = {'build_ext': build_ext},
    ext_modules = ext_modules,
    include_dirs=[np.get_include()]  # This gets all the required Numpy core files
)
```

run.py

`complexFunLib` Python `complexFunLib` DLL .

```
import complexFunLib
import numpy as np

# Create arrays of non-trivial complex numbers to be exponentiated,
# i.e. res = k*exp(ee)
k = np.ones(int(2.5e5), dtype='complex64')*1.1234 + np.complex64(1.1234j)
ee = np.ones(int(2.5e5), dtype='complex64')*1.1234 + np.complex64(1.1234j)
sz = k.size  # Get size integer
res = np.zeros(int(2.5e5), dtype='complex64')  # Create array for results

# Call function
complexFunLib.mp_exp_c4(k, ee, sz, res, 8)

# Print results
print(res)
```

C ++ : https://riptutorial.com/ko/cython/topic/3525/c-plusplus-

3: C

Examples

C

`my_random` C . `set_seed(long seed)` `rand()` (`set_seed(long seed)` . Cython

1. .pxd
2. .pyx .

test_extern.pxd

```
# extern blocks define interfaces for Cython to C code
cdef extern from "my_random.h":
    double rand()
    void c_set_seed "set_seed" (long seed) # rename C version of set_seed to c_set_seed to
avoid naming conflict
```

test_extern.pyx

```
def set_seed (long seed):
    """Pass the seed on to the c version of set_seed in my_random."""
    c_set_seed(seed)

cpdef get_successes (int x, double threshold):
    """Create a list with x results of rand <= threshold

    Use the custom rand function from my_random.
    """
    cdef:
        list successes = []
        int i
    for i in range(x):
        if rand() <= threshold:
            successes.append(True)
        else:
            successes.append(False)
    return successes
```

C : https://riptutorial.com/ko/cython/topic/3626/c--

4: Cython

Examples

pyinstaller Cython

Cython .

```
def do_stuff():
    cdef int a,b,c
    a = 1
    b = 2
    c = 3
    print("Hello World!")
    print([a,b,c])
    input("Press Enter to continue.")
```

`setup.py` .

```
from distutils.core import setup
from Cython.Build import cythonize
setup(
    name = "Hello World",
    ext_modules = cythonize('program.pyx'),
)
```

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

(: `main.py`) Python `.pyd` .

```
import program
program.do_stuff()
```

pyinstaller `pyinstaller --onefile "main.py"` . 4MB .

(Windows)

Windows `.bat` .

```
del "main.exe"
python setup.py build_ext --inplace
del "*.c"
rmdir /s /q ".\build"
pyinstaller --onefile "main.py"
copy /y ".\dist\main.exe" ".\main.exe"
rmdir /s /q ".\dist"
rmdir /s /q ".\build"
del "*.spec"
del "*.pyd"
```

Numpy

Numpy `include_dirs` `setup.py` wrapper Python numpy Pyinstaller .

`program.pyx` :

```
import numpy as np
cimport numpy as np


def do_stuff():
    print("Hello World!")
    cdef int n
    n = 2
    r = np.random.randint(1,5)
    print("A random number: "+str(r))
    print("A random number multiplied by 2 (made by cdef):"+str(r*n))
    input("Press Enter to continue.")
```

`setup.py` :

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

setup(
    ext_modules=cythonize("hello.pyx"),
    include_dirs=[numpy.get_include()]
)
```

`main.py` :

```
import program
import numpy
program.do_stuff()
```

Cython : https://riptutorial.com/ko/cython/topic/6386/cython-

S. No		Contributors
1	Cython	Community, J.J. Hakala, Keith L, m00am
2	C ++	J.J. Hakala, Keith L, Kevin Pasquarella
3	C	m00am
4	Cython	Andrii Magalich