Introduction to Cython - Week 2

Richard Killam

Sunday 4th October, 2015

Outline

Method Overview

There are 3 ways to use Cython files:

1. **Direct Import:** import the code without explicitly compiling

Method Overview

There are 3 ways to use Cython files:

1. **Direct Import:** import the code without <u>explicitly</u> compiling
2. **Compiled Import:** explicitly compile the code, then import

Method Overview

There are 3 ways to use Cython files:

1. **Direct Import:** import the code without <u>explicitly</u> compiling
2. **Compiled Import:** explicitly compile the code, then import
3. **Compiled Executable:** explicitly compile the code and then run it directly

Method Overview

There are 3 ways to use Cython files:

1. **Direct Import:** import the code without explicitly compiling
2. **Compiled Import:** explicitly compile the code, then import
3. **Compiled Executable:** explicitly compile the code and then run it directly

hello.pyx

```
print('Hello World!')
```

Direct Import

```
cd Examples/CythonHelloWorld/DirectImport
ls
python run_hello.py
```

Direct Import

```
cd Examples/CythonHelloWorld/DirectImport
ls
python run_hello.py
```

run_hello.py

```
# pyximport lets you import cython (.pyx) files without
    compiling them first
import pyximport
pyximport.install()

print('Before importing hello')
import hello
print('After importing hello')
```

Compiled Import

```
cd Examples/CythonHelloWorld/CompiledImport
ls
```

setup.py Helper script compiles the given .pyx files into C libraries (.so files)

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

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

Compiled Import cont.

```
python run_hello.py
```

Compiled Import cont.

```
python run_hello.py  # ImportError
```

Compiled Import cont.

```
python run_hello.py  # ImportError
python setup.py build_ext --inplace
ls  # Note hello.so
python run_hello.py
```

run_hello.py

```
print('Before importing hello')
import hello
print('After importing hello')
```

Compiled Import cont.

```
python run_hello.py  # ImportError
python setup.py build_ext --inplace
ls  # Note hello.so
python run_hello.py
```

run_hello.py

```
print('Before importing hello')
import hello
print('After importing hello')
```

Notice the speed difference between Direct and Compiled Importing.

Compiled Executable

```
cd Examples/CythonHelloWorld/CompiledExecutable
ls
```

cython_build.sh Script I wrote to streamline the compilation process.

1. Uses the Cython compiler to compile hello.pyx into hello.c
2. Uses gcc to compile hello.c into an executable

Compiled Executable cont

```
bash cython_build.sh hello.pyx
./hello
```

Compiled Executable cont

```
bash cython_build.sh hello.pyx
./hello
```

Open hello.c

Compiled Executable cont

```
bash cython_build.sh hello.pyx
./hello
```

Open hello.c

```
wc -l hello.c # 1,626 lines!!!
```

Method Summary

- **Direct Import**
  - Slow start-up on each run
  - Simple
  - Good for development

Method Summary

- **Direct Import**
  - Slow start-up on each run
  - Simple
  - Good for development
- **Compiled Import**
  - Fast on each run
  - Added step in execution process
  - Used for production / release

Method Summary

- **Direct Import**
  - Slow start-up on each run
  - Simple
  - Good for development
- **Compiled Import**
  - Fast on each run
  - Added step in execution process
  - Used for production / release
- **Compiled Executable**
  - Complicated compilation process
  - Could be used to develop a module
  - Most used method for this workshop

sum_nums_func.pyx

sum_nums_func.pyx

sum_nums_func.pyx

sum_nums_func.pyx

```
def sum_nums(n):
    s = 0
    for i in range(n+1):
        s += i
    return s

import sys
n = int(sys.argv[1])
print(sum_nums(n))
```

sum_nums_func.pyx

sum_nums_func.pyx

```python
def sum_nums(n):
    s = 0
    for i in range(n+1):
        s += i
    return s

import sys
n = int(sys.argv[1])
print(sum_nums(n))
```

```
time python sum_nums_func.pyx 100000000 # ≈ 14 seconds
cython_build.sh sum_nums_func.pyx
time ./sum_nums_func 100000000 # ≈ 12 seconds
```

Static Type Declaration in Cython

```
cdef char c
cdef unsigned char b
cdef int i
cdef long j
cdef unsigned int k
cdef unsigned long long l
cdef float f
cdef double d
cdef char* s
```

Static Type Declaration in Cython

```
cdef char c
cdef unsigned char b
cdef int i
cdef long j
cdef unsigned int k
cdef unsigned long long l
cdef float f
cdef double d
cdef char* s
cdef struct (Maybe talk about this later)
```

Declaring the Iterator

sum_nums_func.pyx

```
def sum_nums(n):
    s = 0
    cdef unsigned long i # ← defines i as a unsigned long
    for i in range(n+1):
        s += i
    return s

import sys
n = int(sys.argv[1])
print(sum_nums(n))
```

Declaring the Iterator

sum_nums_func.pyx

```
def sum_nums(n):
    s = 0
    cdef unsigned long i # ← defines i as a unsigned long
    for i in range(n+1):
        s += i
     return s

import sys
n = int(sys.argv[1])
print(sum_nums(n))
```

```
time python sum_nums_func.pyx 100000000
```

Declaring the Iterator

sum_nums_func.pyx

```
def sum_nums(n):
    s = 0
    cdef unsigned long i # ← defines i as a unsigned long
    for i in range(n+1):
        s += i
     return s

import sys
n = int(sys.argv[1])
print(sum_nums(n))
```

```
time python sum_nums_func.pyx 100000000 # SyntaxError
```

Declaring the Iterator

sum_nums_func.pyx

```
def sum_nums(n):
    s = 0
    cdef unsigned long i # ← defines i as a unsigned long
    for i in range(n+1):
        s += i
    return s

import sys
n = int(sys.argv[1])
print(sum_nums(n))
```

```
time python sum_nums_func.pyx 100000000 # SyntaxError
cython_build.sh sum_nums_func.pyx
time ./sum_nums_func 100000000 # ≈ 12 seconds (Slightly
    faster than without the cdef)
```

Declaring the Sum

sum_nums_func.pyx

```
def sum_nums(n):
    cdef unsigned long s = 0
    cdef unsigned long i # ← defines i as a unsigned long
    for i in range(n+1):
        s += i
     return s

import sys
n = int(sys.argv[1])
print(sum_nums(n))
```

Declaring the Sum

sum_nums_func.pyx

```
def sum_nums(n):
    cdef unsigned long s = 0
    cdef unsigned long i # ← defines i as a unsigned long
    for i in range(n+1):
        s += i
     return s

import sys
n = int(sys.argv[1])
print(sum_nums(n))
```

```
cython_build.sh sum_nums_func.pyx
time ./sum_nums_func 100000000 # ≈ 0.5 seconds (Slightly
    fasterer than without the cdef)
```

Cython Command & HTML Annotations

```
cython -a --embed ${cython_file} -o ${c_file}
```

-o ${c_file} Specifies the name of the resulting C file

Cython Command & HTML Annotations

```
cython -a --embed ${cython_file} -o ${c_file}
```

-o ${c_file}	Specifies the name of the resulting C file
–embed	Compiles the C code with a main method

Cython Command & HTML Annotations

```
cython -a --embed ${cython_file} -o ${c_file}
```

-o ${c_file}	Specifies the name of the resulting C file
–embed	Compiles the C code with a main method
-a	Produces a helpful HTML file

Cython Command & HTML Annotations

```
cython -a --embed ${cython_file} -o ${c_file}
```

-o ${c_file}	Specifies the name of the resulting C file
–embed	Compiles the C code with a main method
-a	Produces a helpful HTML file

sum_nums_py.html & sum_nums_cy.html

Return and Parameter Typing

sum_nums_func.pyx

```
cdef sum_nums(unsigned long n):
    cdef unsigned long s = 0
    cdef unsigned long i
    for i in range(n+1):
        s += i
     return s

import sys
n = int(sys.argv[1])
print(sum_nums(n))
```