Introduction to Cython - Week 2

Richard Killam

Sunday 1st November, 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 explicitly 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 explicitly 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

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

Direct Import


```
1       cd Examples/CythonHelloWorld/DirectImport
2       ls
3       python run_hello.py
```

Direct Import

```
1      cd Examples/CythonHelloWorld/DirectImport
2      ls
3      python run_hello.py
```

run_hello.py

```
1  # pyximport lets you import cython (.pyx) files without
       compiling them first
2  import pyximport
3  pyximport.install()
4
5  print('Before importing hello')
6  import hello
7  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.

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

Compiled Import cont.

```
1     python run_hello.py  # ImportError
2     python setup.py build_ext --inplace
3     ls  # Note hello.so
4     python run_hello.py
```

run_hello.py

```
1 print('Before importing hello')
2 import hello
3 print('After importing hello')
```

Compiled Import cont.

```
1     python run_hello.py  # ImportError
2     python setup.py build_ext --inplace
3     ls  # Note hello.so
4     python run_hello.py
```

run_hello.py

```
1 print('Before importing hello')
2 import hello
3 print('After importing hello')
```

Notice the speed difference between Direct and Compiled Importing.

Compiled Executable

```
1       cd Examples/CythonHelloWorld/CompiledExecutable
2       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

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

Compiled Executable cont

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

Open hello.c

Compiled Executable cont

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

Open hello.c

```
1    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

```
1       def sum_nums(n):
2           s = 0
3           for i in range(n+1):
4               s += i
5           return s
6
7       import sys
8       n = int(sys.argv[1])
9       print(sum_nums(n))
```

sum_nums_func.pyx

sum_nums_func.pyx

```
1   def sum_nums(n):
2       s = 0
3       for i in range(n+1):
4           s += i
5       return s
6
7   import sys
8   n = int(sys.argv[1])
9   print(sum_nums(n))
```

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

Static Type Declaration in Cython


```
1       cdef char c
2       cdef unsigned char b
3       cdef int i
4       cdef long j
5       cdef unsigned int k
6       cdef unsigned long long l
7       cdef float f
8       cdef double d
9       cdef char* s
```

Static Type Declaration in Cython


```
1     cdef char c
2     cdef unsigned char b
3     cdef int i
4     cdef long j
5     cdef unsigned int k
6     cdef unsigned long long l
7     cdef float f
8     cdef double d
9     cdef char* s
10    cdef struct (Maybe talk about this later)
```

Declaring the Iterator

sum_nums_func.pyx

```
1   def sum_nums(n):
2       s = 0
3       cdef unsigned long i # ← defines i as a unsigned long
4       for i in range(n+1):
5           s += i
6       return s
7
8   import sys
9   n = int(sys.argv[1])
10  print(sum_nums(n))
```

Declaring the Iterator

sum_nums_func.pyx

```
1     def sum_nums(n):
2         s = 0
3         cdef unsigned long i # ← defines i as a unsigned long
4         for i in range(n+1):
5               s += i
6           return s
7
8     import sys
9     n = int(sys.argv[1])
10    print(sum_nums(n))
```

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

Declaring the Iterator

sum_nums_func.pyx

```
1   def sum_nums(n):
2       s = 0
3       cdef unsigned long i # ← defines i as a unsigned long
4       for i in range(n+1):
5           s += i
6       return s
7
8   import sys
9   n = int(sys.argv[1])
10  print(sum_nums(n))
```

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

Declaring the Iterator

sum_nums_func.pyx

```
1   def sum_nums(n):
2       s = 0
3       cdef unsigned long i # ← defines i as a unsigned long
4       for i in range(n+1):
5           s += i
6       return s
7
8   import sys
9   n = int(sys.argv[1])
10  print(sum_nums(n))
```

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

Declaring the Sum

sum_nums_func.pyx

```
1   def sum_nums(n):
2       cdef unsigned long s = 0
3       cdef unsigned long i # ← defines i as a unsigned
            long
4       for i in range(n+1):
5           s += i
6       return s
7
8   import sys
9   n = int(sys.argv[1])
10  print(sum_nums(n))
```

Declaring the Sum

sum_nums_func.pyx

```
1   def sum_nums(n):
2       cdef unsigned long s = 0
3       cdef unsigned long i # ← defines i as a unsigned
            long
4       for i in range(n+1):
5           s += i
6       return s
7
8   import sys
9   n = int(sys.argv[1])
10  print(sum_nums(n))
```

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

Cython Command & HTML Annotations

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

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

Cython Command & HTML Annotations

```
1       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

```
1    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

```
1    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

```
1   cdef sum_nums(unsigned long n):
2       cdef unsigned long s = 0
3       cdef unsigned long i
4       for i in range(n+1):
5           s += i
6       return s
7
8   import sys
9   n = int(sys.argv[1])
10  print(sum_nums(n))
```