# Introduction to Cython - Week 2

Richard Killam

Sunday 4th October, 2015

# Outline

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. **Compile Executable:** explicitly compile the code and then run it directly

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. **Compile 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
```

run_hello.py

```
1  import pyximport
2  pyximport.install()
3
4  print('Before importing hello')
5  import hello
6  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.

```
1 python run_hello.py
```

# 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 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 hello.c into executable

# Compiled Executable cont

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

# Getting numbers in

name_and_age.py

```
1  name = raw_input("Name > ")
2  year_of_birth = int(raw_input("Year of Birth > "))
3
4  print("{NAME} is {YEARS} old.".format(
5      NAME=name,
6      YEARS=2015 - year_of_birth
7  ))
```

# if, else

True_False.py

```
1  if True:
2      print("The conditional was True")
3
4  else:
5      print("The conditional was False")
```

# if, else

True_False.py

```
1  if True:
2      print("The conditional was True")
3
4  else:
5      print("The conditional was False")
```

Note the indentation.

# if, else with input

even_or_odd.py

```
1  num = int(raw_input("Number > "))
2
3  if num % 2 == 0:
4      print("{NUM} is even".format(NUM=num))
5
6  else:
7      print("{NUM} is odd".format(NUM=num))
```

# elif

a_or_b.py

```
1  choice = raw_input("Choose a or b ")
2
3  if choice == "a":
4      print("You chose a")
5
6  elif choice == "b":
7      print("You chose b")
8
9  else:
10     print("You did not follow instructions...")
```

# While + and

force_a_or_b.py

```
1  choice = ""
2
3  while choice != "a" and choice != "b":
4  choice = raw_input("Choose a or b >")
5
6  if choice == "a":
7      print("You chose a")
8  elif choice == "b":
9      print("You chose b")
10 else:
11     # Dead code
12     print("How did you get here?!?!")
```

# While + or + not

force_a_or_b.py

```
1  choice = ""
2
3  while not (choice == "a" or choice == "b"):
4  choice = raw_input("Choose a or b >")
5
6  if choice == "a":
7      print("You chose a")
8  elif choice == "b":
9      print("You chose b")
10 else:
11     # Dead code
12     print("How did you get here?!?!")
```

# Array declaration

array_declaration.py

```
1  a = [1, 2, 3, 4, 5]
2  print(a) # >> [1, 2, 3, 4, 5]
3
4  a = range(1, 6)
5  print(a) # >> [1, 2, 3, 4, 5]
6
7  a = range(5)
8  print(a) # >> [0, 1, 2, 3, 4]
```

# Array indexing

array_indexing.py

```
1  a = range(5)
2
3  print(a) # >> [0, 1, 2, 3, 4]
4  print(a[0]) # >> 0
5
6  print(a[len(a) - 1]) # >> 4
7  print(a[-1]) # >> 4
8
9  print(a[len(a) - 2]) # >> 3
10 print(a[-2]) # >> 3
```

# C-Style for loop

c_style_for_loop.py

```
1  a = range(5, 10)
2
3  for i in range(len(a)):
4      print("a[{I}] = {AI}".format(I=i, AI=a[i]))
```

# Python-Style for loop

python_style_for_loop.py

```
1  a = range(5, 10)
2
3  for ai in a:
4      print("AI = {AI}".format(AI=a[i]))
5
```

# Python-Style for loop

python_style_for_loop.py

```
1  a = range(5, 10)
2
3  for ai in a:
4      print("AI = {AI}".format(AI=a[i]))
5
6  for i, ai in enumerate(a):
7      print("a[{I}] = {AI}".format(I=i, AI=ai))
```

# Exercise - Sum the Numbers from 1 to n

sum_nums.py

# Exercise - Sum the Numbers from 1 to n

sum_nums.py

```
1  n = int(raw_input("n (-1 to quit) > "))
2  while n > 1:
3      s = 0
4
5      # n + 1 because range goes to n - 1
6      for i in range(1, n+1):
7          s += i
8
9       print("n = {N} -> {S}".format(N=n, S=s))
10
11     n = int(raw_input("n (-1 to quit) > "))
```

# def

sum_nums_func.py

```
1  def sum_nums(n):
2      s = 0
3
4      # n + 1 because range goes to n - 1
5      for i in range(1, n+1):
6          s += i
7
8      return s
9
10 print("sum_nums({N}) => {SNN}".format(
11     N=10,
12     SNN=sum_nums(10)
13 ))  # >> 55
```

# Default Arguments

func_default_args.py

```
1  def default_args(a=1, b=3):
2      print("a was {A}; b was {B}".format(A=a, B=b))
3
4  default_args()            # >> a was 1; b was 3
5  default_args(7)           # >> a was 7; b was 3
6  default_args(7, 11)       # >> a was 7; b was 11
7  default_args(7, b=12)     # >> a was 7; b was 12
8  default_args(a=8, b=12)   # >> a was 8; b was 12
9  default_args(b=13, a=5)   # >> a was 5; b was 13
10 default_args(a=10, 14)    # >> SyntaxError: non-keyword arg
       after keyword arg
```

# Exercise - Sum the Numbers from low to high

sum_range.py

# Exercise - Sum the Numbers from low to high

sum_range.py

```
1  def sum_range(low=1, high=10):
2      s = 0
3
4      # high + 1 because range goes to high - 1
5      for i in range(low, high + 1):
6          s += i
7
8      return s
```

# Basics

basic_import_from_python_path.py

```
1  # imports that math library and references it as math
2  import math
3  print(math.log(100, 10)) # >> 2.0 (log 100 base 10)
4
5  # sys.argv allows for command line arguments
6  import sys
7  print(sys.argv)
```

# Different styles of importing

```
1  # imports the sparse module from the scipy library as
       scipy.sparse
2  import scipy.sparse
3
4  # imports the sparse module from the scipy library as
       sp_sparse
5  import scipy.sparse as sp_sparse
6
7  # imports the sparse module from the scipy library as
       sparse
8  from scipy import sparse
```

# From sum_range.py

import_sum_range.py

```
1  # This is the only style that I will use in the workshop
2  import sum_range # Note that there is no .py
3  print(sum_range.sum_range(low=2, high=10)) # >> 54
4
5  import sum_range as sr
6  print(sr.sum_range(low=2, high=10)) # >> 54
7
8  from sum_range import sum_range
9  print(sum_range(low=2, high=10)) # >> 54
10
11 from sum_range import sum_range as sr
12 print(sr(low=2, high=10)) # >> 54
```

# Definition, Instantiation, and Usage

animal.py

```
1  # Explicit inheritance from object class
2  class Animal(object):
3      # Explicit (and necessary) passing of self object
4      def __init__(self, name_in, noise_in):
5          self.name = name_in #  ≠  name = name_in
6          self.noise = noise_in #  ≠  noise = noise_in
7
8  dog = Animal("Rex", "woof")
9  print("{DOG} makes a {NOISE} noise".format(
10     DOG=dog.name,
11     NOISE=dog.noise
12 ))
```

# Inheritance

animal.py

```
1  class Dog(Animal):
2      def __init__(self, name):
3          # This gets better in Python 3
4          super(Dog, self).__init__(name, "woof")
5
6      # NEED to accept self
7      def wag_tail(self):
8           print("{NAME} is happy".format(NAME=self.name))
9
10  rex = Dog("Rex")
11  rex.wag_tail() # DON'T need to explicitly pass self
```