Cython vs. SWIG, Fight!

Mark Kohler

2013-03-16

Wrapping C libraries for Python

Pre-fight

- import statement
- libadder
- passing ints
- passing structs
- C strings
- memory management
- generalizations

Code first and ask questions later

1. C code
2. SWIG code
3. SWIG demonstration
4. Cython code
5. Cython demonstration

We are Here

- **import statement**
- libadder
- passing ints
- passing structs
- C strings
- memory management
- generalizations

import this

```
>>> import socket
>>> import datetime
>>> import time
```

- What is socket.__file__?
- What is datetime.__file__?
- What is time.__file__?

import socket

```
>>> socket.__file__
'/usr/lib/python2.7/socket.pyc'
>>>
```

import datetime

```
>>> datetime.__file__
'/usr/lib/python2.7/lib-dynload/datetime.so'
>>>

$ file /usr/lib/python2.7/lib-dynload/datetime.so
ELF 64-bit LSB shared object, x86-64, dynamically linked
$
```

import time

```
>>> time.__file__
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
AttributeError: 'module' object has
no attribute '__file__'
>>>
```

We are Here

- import statement
- **libadder**
- passing ints
- passing structs
- C strings
- memory management
- generalizations

adder.c: add()

```
int
add(int x, int y) {
    return x + y;
}
```

adder.h: add()

```
int add(int x, int y);
```

Building libadder.so

1. Compile
   adder.c + adder.h --> adder.o
2. Link
   adder.o --> libadder.so

We are Here

- import statement
- libadder
- **passing ints**
- passing structs
- C strings
- memory management
- generalizations

adder.i (SWIG interface file)

```
%module adder
%{
#include "adder.h"
%}

int add(int, int);
```

SWIG build diagram

1. Start with: **adder.h**, libadder.so
2. SWIG
   adder.h + adder.i --> adder_wrap.c + adder.py
3. Compile
   adder.h + adder-wrap.c --> adder_wrap.o
4. Link
   libadder.so + adder_wrap.o --> _adder.so

demo of SWIG's add()

```
>>> import adder
>>> adder.add(2, 3)
5
>>>
```

c_adder.pxd: Cython interface file

```
cdef extern from "adder.h":
    int add(int x, int y)
```

cy_adder.pyx: Cython source file

```
cimport c_adder

def add(x, y):
    return c_adder.add(x, y)
```

Cython build diagram

1. Cython
   adder.h + c_adder.pxd + cy_adder.pyx -->
   cy_adder.c
2. Compile
   adder.h + cy_adder.c --> cy_adder.o
3. Link
   lib_adder.so + cy_adder.o --> cy_adder.so

demo of Cython's add()

```
>>> import cy_adder
>>> cy_adder.add(2, 3)
5
>>>
```

Cython build review

1. Given: adder.h, libadder.so
2. Cython
   adder.h + c_adder.pxd + cy_adder.pyx -->
   cy_adder.c
3. Compile
   adder.h + cy_adder.c --> cy_adder.o
4. Link
   libadder.so + cy_adder.o --> cy_adder.so

We are Here

- import statement
- libadder
- passing ints
- **passing structs**
- C strings
- memory management
- generalizations

adder.h: pair_add()

```
typedef struct _PAIR {
    int x;
    int y;
} PAIR;

int pair_add(PAIR * ppair);
```

adder.c: pair_add()

```
int
pair_add(PAIR * ppair) {
    return ppair->x + ppair->y;
}
```

adder.i: pair_add()

```
typedef struct _PAIR {
    int x;
    int y;
} PAIR;

int pair_add(PAIR * ppair);
```

demo of SWIG's pair_add()

```
>>> import adder
>>> my_pair = adder.PAIR()
>>> type(my_pair)
<class 'adder.PAIR'>
>>> my_pair.x = 3
>>> my_pair.y = 4
>>> adder.pair_add(my_pair)
7
>>>
```

c_adder.pxd: pair_add()

```
ctypedef struct PAIR:
    int x
    int y

int pair_add(PAIR * ppair)
```

cy_adder.pyx: pair_add()

```
def pair_add(x, y):
  cdef c_adder.PAIR my_pair
  my_pair.x = x
  my_pair.y = y
  return c_adder.pair_add(&my_pair)
```

demo of Cython's pair_add()

```
>>> import cy_adder
>>> cy_adder.pair_add(3, 4)
7
>>>
```

We are Here

- import statement
- libadder
- passing ints
- passing structs
- **C strings**
- memory management
- generalizations

adder.h: get_version()

```
char * get_version(void);
```

adder.c: get_version()

```
static char version[] = "v1.0";

char *
get_version(void) {
    return version;
}
```

adder.i: get_version()

```
char * get_version(void);
```

demo of SWIG's get_version()

```
>>> import adder
>>> adder.get_version()
'v1.0'
>>> adder.get_version().__class__
<type 'str'>
>>>
```

c_adder.pxd: get_version()

```
char * get_version()
```

cy_adder.pyx: get_version()

```
def get_version():
    return c_adder.get_version()
```

demo of Cython's get_version()

```
>>> import cy_adder
>>> cy_adder.get_version()
'v1.0'
>>> cy_adder.get_version().__class__
<type 'str'>
>>>
```

Cython and C Strings

"C strings are slow and cumbersome"
"...avoid using C strings where possible"
"...more likely to introduce bugs"

SWIG and C Strings

"The problems (and perils) of using char * are well-known. However, SWIG is not in the business of enforcing morality."

SWIG documentation, Section 8.3 C String Handling

We are Here

- import statement
- libadder
- passing ints
- passing structs
- C strings
- **memory management**
- generalizations

adder.h: sgreeting()

```
int sgreeting(char * name,
              char * outp,
              int buflen);
```

adder.c: sgreeting()

```
static char hello[] = "Hello, ";

int
sgreeting(char * name, char * outp, int buflen) {
    if (outp && buflen) {
        if (buflen < (strlen(hello) +
                      strlen(name) + 1)) {
            outp[0] = 0;
            return 0;
        }
        strcpy(outp, hello);
        strcat(outp, name);
    }
    return strlen(hello) + strlen(name);
}
```

adder.i: sgreeting()

```
%include "cstring.i"
%cstring_output_maxsize(char * outp,
                        int buflen);

int sgreeting(char * name,
              char * outp,
              int buflen);
```

demo of SWIG's sgreeting()

```
>>> import adder
>>> adder.sgreeting("Monty", 100)
[12, 'Hello, Monty']
>>>
```

c_adder.pxd: sgreeting()

```
int sgreeting(char * name,
              char * output,
              int buflen)
```

cy_adder.pyx: sgreeting()

```
def sgreeting(name):
    c_len = c_adder.sgreeting(name, <char * > 0, 0)
    py_str = 'x' * (c_len + 1)
    cdef char * c_str = py_str
    c_adder.sgreeting(name, c_str, len(py_str))
    return c_str
```

demo of Cython's sgreeting()

```
>>> import cy_adder
>>> cy_adder.sgreeting("Monty")
'Hello, Monty'
>>>
```

We are Here

- import statement
- libadder
- passing ints
- passing structs
- C strings
- memory management
- **generalizations**

SWIG Advantages

- Multi-language support
- More DRY than Cython

Cython Advantages

- It's Python **and** it's C
- explore performance trade-offs between C and Python

Alternatives to Cython and SWIG

- Python C/API
  http://docs.python.org/2/extending/
- ctypes

Getting Started

- Start small
- Use distutils

Cython vs. SWIG, Vote

Code and Slides

https://github.com/mkohler/cython_swig

mark.kohler@gmail.com