Interfacing Python and C using Ctypes

Giuseppe Piero Brandino
and
Jimmy Aguilar Mena

March 9, 2016

Motivation

When do we want to interface Python and C?

- To extend Python functionality

Motivation

When do we want to interface Python and C?

- To extend Python functionality
- To improve performance

Motivation

When do we want to interface Python and C?

- To extend Python functionality
- To improve performance
- To use Python as a glue language

Motivation

When do we want to interface Python and C?

- To extend Python functionality
- To improve performance
- To use Python as a glue language
- To create Python bindings for a library

Example 1: C vs Python

add_numbers.c

```c
#include <stdio.h>

int main(int argc, char **argv){

  int i, j, total;
  double avg;
  total = 10000000;
  for (i = 0; i < 10; i++){
        avg = 0;
        for (j = 0; j < total; j++){
          avg += j;
          }
        avg = avg/total;
        }
  printf("Average is %f\n", avg);
  return 0;
  }
```

add_numbers.py

```python
total = 10000000

for i in xrange(10):
  avg = 0.0
  for j in xrange(total):
        avg += j
  avg = avg/total

print "Average is {0}".format(avg)
```

Compile and execute

gcc -O3 add_numbers.c -o
add_numbers.x time
./add_numbers.x

Execute the Python script

time ./add_numbers.py

Example 1: C vs Python

add_numbers.c

```c
#include <stdio.h>

int main(int argc, char **argv){

  int i, j, total;
  double avg;
  total = 10000000;
  for (i = 0; i < 10; i++){
        avg = 0;
        for (j = 0; j < total; j++){
          avg += j;
          }
        avg = avg/total;
        }
  printf("Average is %f\n", avg);
  return 0;
  }
```

Compile and execute

```
gcc -O3 add_numbers.c -o
add_numbers.x time
./add_numbers.x
```

add_numbers.py

```python
total = 10000000

for i in xrange(10):
  avg = 0.0
  for j in xrange(total):
        avg += j
  avg = avg/total

print "Average is {0}".format(avg)
```

Execute the Python script

```
time ./add_numbers.py
```

Program	time
Python:	20.17s
C:	0.09s

Yes, C is so much faster!!!

Example 1: Sometimes you can use Numpy

add_numbers_np.c

```
from numpy import mean, arange

total = 10000000

a = arange(total)

for i in xrange(10):
        avg = mean(a)
print "Average is {0}".format(avg)
```

Program	time
Python:	20.17s
C:	0.09s
Numpy:	0.17s

Example 1: Sometimes you can use Numpy

add_numbers_np.c

```
from numpy import mean, arange

total = 10000000

a = arange(total)

for i in xrange(10):
        avg = mean(a)
print "Average is {0}".format(avg)
```

Program	time
Python:	20.17s
C:	0.09s
Numpy:	0.17s

Numpy is almost as fast as C because it is written in FORTRAN

Integrating python with other languages

Python can be interfaced with almost any popular language see:
https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages

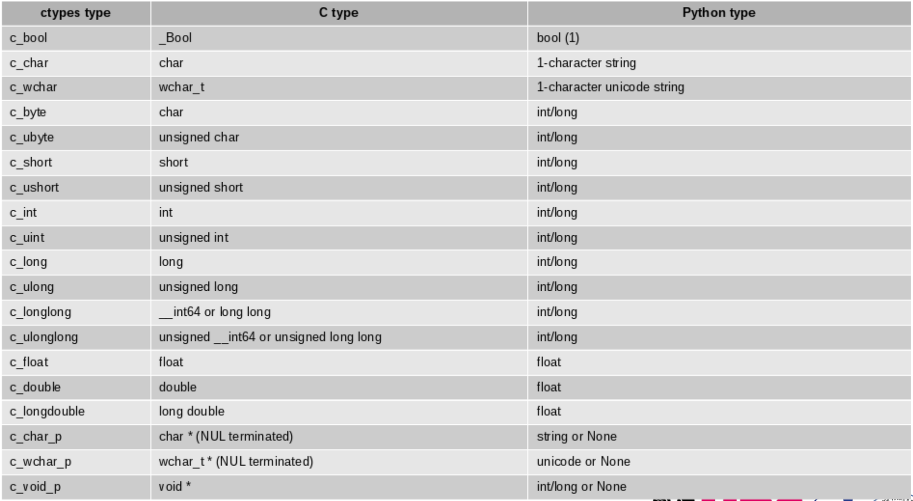
There are too many options for C

- Python C API
- Ctypes
- Cython
- Boost
- Swig
- pybind11

Ctypes

- Ctypes is a foreign function library for Python.
- It provides C compatible data types, and allows calling functions in DLLs or shared libraries.

Ctypes types and C types



ctypes type	C type	Python type
c_bool	_Bool	bool (1)
c_char	char	1-character string
c_wchar	wchar_t	1-character unicode string
c_byte	char	int/long
c_ubyte	unsigned char	int/long
c_short	short	int/long
c_ushort	unsigned short	int/long
c_int	int	int/long
c_uint	unsigned int	int/long
c_long	long	int/long
c_ulong	unsigned long	int/long
c_longlong	__int64 or long long	int/long
c_ulonglong	unsigned __int64 or unsigned long long	int/long
c_float	float	float
c_double	double	float
c_longdouble	long double	float
c_char_p	char * (NUL terminated)	string or None
c_wchar_p	wchar_t * (NUL terminated)	unicode or None
c_void_p	void *	int/long or None

Example 2: Library

example2/add.c

```c
float add_float(float a,float b){return a + b;}

int add_int(int a,int b){return a + b;}

int add_float_ref(float *a,float *b,float *c){
  *c = *a + *b;
  return 0;
  }
```

example2/arrays.c

```c
int add_int_array(int *a,int *b,int *c, int n){
  int i;
  for (i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
    }
  return 0;
  }
float dot_product(float *a, float *b, int n) {
  float res=0;
  int i;
  for (i = 0; i < n; i++) {
    res = res + a[i] * b[i];
    }
  return res;
  }
```

Compile and create the library

```
$ gcc -fPIC -c add.c
$ gcc -fPIC -c arrays.c

$ gcc -shared add.o arrays.o -o

libmymath.so
```

Example 2: Library

example2/add.c

```c
float add_float(float a,float b){return a + b;}

int add_int(int a,int b){return a + b;}

int add_float_ref(float *a,float *b,float *c){
  *c = *a + *b;
  return 0;
  }
```

example2/arrays.c

```c
int add_int_array(int *a,int *b,int *c, int n){
  int i;
  for (i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
    }
  return 0;
  }
float dot_product(float *a, float *b, int n) {
  float res=0;
  int i;
  for (i = 0; i < n; i++) {
    res = res + a[i] * b[i];
    }
  return res;
  }
```

Compile and create the library

```
$ gcc -fPIC -c add.c
$ gcc -fPIC -c arrays.c

$ gcc -shared add.o arrays.o -o

libmymath.so
```

In a python interpreter

```python
import ctypes
math= ctypes.CDLL("libmymath.so")
math.add_int(4,5)
```

Example 2: Library

example2/add.c

```c
float add_float(float a,float b){return a + b;}

int add_int(int a,int b){return a + b;}

int add_float_ref(float *a,float *b,float *c){
  *c = *a + *b;
  return 0;
  }
```

example2/arrays.c

```c
int add_int_array(int *a,int *b,int *c, int n){
  int i;
  for (i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
    }
  return 0;
  }
float dot_product(float *a, float *b, int n) {
  float res=0;
  int i;
  for (i = 0; i < n; i++) {
    res = res + a[i] * b[i];
    }
  return res;
  }
```

Compile and create the library

```
$ gcc -fPIC -c add.c
$ gcc -fPIC -c arrays.c

$ gcc -shared add.o arrays.o -o

libmymath.so
```

In a python interpreter

```python
import ctypes
math= ctypes.CDLL("libmymath.so")
math.add_int(4,5)
```

Example 2: Arguments

But this:

```
math.add_float(4,5)
math.add_float(4.0,5.0)
```

Example 2: Arguments

But this:

```
math.add_float(4,5)
math.add_float(4.0,5.0)
```

Will produce an error:

```
ArgumentError            Traceback (most recent call last)
<ipython-input-9-a461a3162c94> in <module> ()
– – – – – > math.add_float ( 4.0 , 5.0 )
ArgumentError : argument 1: <class 'TypeError'>: Don't know how to convert parameter 1
```

Example 2: Arguments

But this:

```
math.add_float(4,5)
math.add_float(4.0,5.0)
```

We need to specify the correct type for the arguments and the return type:

```
math.add_float.restype=ctypes.c_float
math.add_float(ctypes.c_float(4.0),ctypes.c_float(5.0))
```

Example 2: Arguments

But this:

```
math.add_float(4,5)
math.add_float(4.0,5.0)
```

We need to specify the correct type for the arguments and the return type:

```
math.add_float.restype=ctypes.c_float
math.add_float(ctypes.c_float(4.0),ctypes.c_float(5.0))
```

We can also specify argument types once and for all, using argtypes

```
math.add_float.restype=ctypes.c_float
math.add_float.argtypes= [ctypes.c_float, ctypes.c_float]
math.add_float(4.0,5.0)
```

Example 2: Passing by reference

Specifying the parameters:

```
a=ctypes.c_float(5)
b=ctypes.c_float(5)
res=ctypes.c_float()

math.add_float_ref(ctypes.byref(a),
                   ctypes.byref(b),
                   ctypes.byref(res))
res.value
```

Example 2: Passing by reference

Specifying the parameters:

```
a=ctypes.c_float(5)
b=ctypes.c_float(5)
res=ctypes.c_float()

math.add_float_ref(ctypes.byref(a),
                   ctypes.byref(b),
                   ctypes.byref(res))
res.value
```

We can also use ctypes.pointer

```
a=ctypes.c_float(5)
b=ctypes.c_float(5)
res=ctypes.c_float()

i=ctypes.pointer(a)
j=ctypes.pointer(b)
k=ctypes.pointer(res)
```

```
math.add_float_ref(i,j,k)
res.value
k.contents
```

Example 2: Passing by reference

Specifying the parameters:

```
a=ctypes.c_float(5)
b=ctypes.c_float(5)
res=ctypes.c_float()

math.add_float_ref(ctypes.byref(a),
                   ctypes.byref(b),
                   ctypes.byref(res))
res.value
```

We can also use ctypes.pointer

```
a=ctypes.c_float(5)
b=ctypes.c_float(5)
res=ctypes.c_float()

i=ctypes.pointer(a)
j=ctypes.pointer(b)
k=ctypes.pointer(res)
```

```
math.add_float_ref(i,j,k)
res.value
k.contents
```

Example 2: Arrays (pure Ctypes)

```
a=(ctypes.c_int * 3) (-1, 2, 5)
b=(ctypes.c_int * 3) (-1, 3, 3)
res=(ctypes.c_int * 3) (0, 0, 0)
n=ctypes.c_int(3)

math.add_int_array(a,b, res,n)

res[0], res[1], res[2]
```

Example 2: Arrays (pure Ctypes)

```
a=(ctypes.c_int * 3) (-1, 2, 5)
b=(ctypes.c_int * 3) (-1, 3, 3)
res=(ctypes.c_int * 3) (0, 0, 0)
n=ctypes.c_int(3)

math.add_int_array(a,b, res,n)

res[0], res[1], res[2]
```

Default ctypes way of creating arrays

Example 2: Arrays (using Numpy)

```
import numpy as np
a=np.array([1,2,-5], dtype=ctypes.c_int)
b=np.array([-1,3,3], dtype=ctypes.c_int)
res= np.zeros(3, dtype=ctypes.c_int)
n=ctypes.c_int(3)
intp=ctypes.POINTER(ctypes.c_int)

i=a.ctypes.data_as(intp)
j=b.ctypes.data_as(intp)
k=res.ctypes.data_as(intp)
math.add_int_array(i,j,k,n)

res
```

Example 2: Arrays (using Numpy)

```
import numpy as np
a=np.array([1,2,-5], dtype=ctypes.c_int)
b=np.array([-1,3,3], dtype=ctypes.c_int)
res= np.zeros(3, dtype=ctypes.c_int)
n=ctypes.c_int(3)
intp=ctypes.POINTER(ctypes.c_int)

i=a.ctypes.data_as(intp)
j=b.ctypes.data_as(intp)
k=res.ctypes.data_as(intp)
math.add_int_array(i,j,k,n)

res
```

We declare the *pointer to int* type as an object

Example 2: Arrays (using Numpy)

```python
import numpy as np
a=np.array([1,2,-5], dtype=ctypes.c_int)
b=np.array([-1,3,3], dtype=ctypes.c_int)
res= np.zeros(3, dtype=ctypes.c_int)
n=ctypes.c_int(3)
intp=ctypes.POINTER(ctypes.c_int)

i=a.ctypes.data_as(intp)
j=b.ctypes.data_as(intp)
k=res.ctypes.data_as(intp)
math.add_int_array(i,j,k,n)

res
```

Ctypes objects are structs with a pointer to an array called *data*.

Structures

example2/rectangle.c

```c
typedef struct _rect {
  float height, width;
  } Rectangle;

float area(Rectangle rect){
  return rect.height*rect.width
  }
```

```
gcc -fPIC -c rectangle.c
gcc -shared rectangle.o -o libgeom.so
```

```python
from geometry import *

r= Rectangle(3,4)

r.area()
r.width=10
r.area()
```

example2/geometry.py

```python
import ctypes as C
clib = C.CDLL('./libgeom.so')
clib.area.argtypes=[C.Structure]
clib.area.restype=C.c_float

class Rectangle(C.Structure):
  _fields_=[
            ("width",C.c_float),
            ("height",C.c_float)
            ]

  def __init__(self,width,height)
    self.width = width
    self.height = height

  def area(self):
    return clib.area(self)
```

Structures

example2/rectangle.c

```
typedef struct _rect {
  float height, width;
  } Rectangle;

float area(Rectangle rect){
  return rect.height*rect.width
  }
```

```
gcc -fPIC -c rectangle.c
gcc -shared rectangle.o -o libgeom.so
```

```
from geometry import *

r= Rectangle(3,4)

r.area()
r.width=10
r.area()
```

example2/geometry.py

```
import ctypes as C
clib = C.CDLL('./libgeom.so')
clib.area.argtypes=[C.Structure]
clib.area.restype=C.c_float

class Rectangle(C.Structure):
  _fields_=[
            ("width",C.c_float),
            ("height",C.c_float)
            ]

  def __init__(self,width,height)
    self.width = width
    self.height = height

  def area(self):
    return clib.area(self)
```

Master in High Performance Computing

SISSA

Structures

example2/rectangle.c

```c
typedef struct _rect {
  float height, width;
  } Rectangle;

float area(Rectangle rect){
  return rect.height*rect.width
  }
```

```
gcc -fPIC -c rectangle.c
gcc -shared rectangle.o -o libgeom.so
```

```python
from geometry import *

r= Rectangle(3,4)

r.area()
r.width=10
r.area()
```

example2/geometry.py

```python
import ctypes as C
clib = C.CDLL('./libgeom.so')
clib.area.argtypes=[C.Structure]
clib.area.restype=C.c_float

class Rectangle(C.Structure):
 _fields_=[
           ("width",C.c_float),
           ("height",C.c_float)
           ]

  def __init__(self,width,height)
    self.width = width
    self.height = height

  def area(self):
    return clib.area(self)
```

Structures

example2/rectangle.c

```c
typedef struct _rect {
  float height, width;
  } Rectangle;

float area(Rectangle rect){
  return rect.height*rect.width
  }
```

```
gcc -fPIC -c rectangle.c
gcc -shared rectangle.o -o libgeom.so
```

```python
from geometry import *

r= Rectangle(3,4)

r.area()
r.width=10
r.area()
```

example2/geometry.py

```python
import ctypes as C
clib = C.CDLL('./libgeom.so')
clib.area.argtypes=[C.Structure]
clib.area.restype=C.c_float

class Rectangle(C.Structure):
 _fields_=[
           ("width",C.c_float),
           ("height",C.c_float)
           ]

  def __init__(self,width,height)
    self.width = width
    self.height = height

  def area(self):
    return clib.area(self)
```