Python programming — Interfacing with other languages

Finn Årup Nielsen

DTU Compute
Technical University of Denmark

September 2, 2013

Overview

Hello-world examples with different approaches.

Calling C(++): Boost.Python,

ctypes

Cython

Other ways to make Python faster

Embedding Python in another language

Calling C et al.

You can call C, C++ and Fortran functions from Python:

Either with "manual" wrapping

Or by using a automated wrapper: SWIG, Boost.Python, CFFI.

or by direct calling existing libraries via `ctypes`.

You can make C-programs in Python with Cython or Pyrex and calling compiled modules from Python

Why calling C et al.?

Why calling C et al.?

- Because you already have code in that language.
- Because ordinary Python is not fast enough.

Boost.Python

Boost is a collection of quality C++ libraries and Boost.Python is one of the libraries.

Boost.Python allows interoperability between C++ and Python

Boost.Python is available on some Linux distributions, e.g., may be installed on Ubuntu with (`sudo aptitude install libboost-python-dev`)

Calling C++ from Python via boost

Hello World example from Boost.Python tutorial with `hello.cpp`:

```
#include <boost/python.hpp>

char const* greet() { return "hello, world"; }  # An "ordinary"
                                                # C function
BOOST_PYTHON_MODULE(hello) {
    using namespace boost::python;
    def("greet", greet); }
```

Once compiled it allows you to call `greet` in the `hello` module:

```
>>> import hello
>>> hello.greet()
'hello, world'
```

Boost building

Compiling and linking on a Linux-like environment:

```
g++ -I/usr/include/python2.7 -fPIC -c hello.cpp
g++ -shared hello.o -lboost_python -o hello.so
```

This produces the `hello.so` shared library (on Linux) that is the module available to python as “hello”.

The greet symbol is somewhere in the `hello.so` shared library:

```
$ nm hello.so | grep greet
0000000000004f9c T _Z5greetv
```

. . . Boost.Python

Note that the code could be split in two files:

A file with ordinary C/C++ declarations/definitions (`greeter.cpp`):

```
char const* greet()
{
  return "hello, world";
}
```

and a Boost.Python file (`hello.cpp`):

```
#include <boost/python.hpp>

#include "greeter.cpp"

BOOST_PYTHON_MODULE(hello) {
    using namespace boost::python;
    def("greet", greet); }
```

SWIG

SWIG (Simplified Wrapper Interface Generator): same target as Boost.Python, i.e., automatically creating a wrapper around c-code.

See also (Langtangen, 2008, chapter 5) Combining Python with Fortran, C, and C++

ctypes

ctypes = call C functions in existing libraries “directly”.

ctypes

Get access to functions in a shared library here `hello.so`

```
gcc -shared hello.c -fPIC -o hello.so
```

constructed from `hello.c` that contains a greet function

```
char const* greet() { return "hello, world"; }
```

In Python you import the `ctypes` library:

```
>>> from ctypes import *
>>> hello = CDLL('hello.so')
>>> hello.greet()
-1068526138
```

Oops wrong return argument!?

. . . ctypes

It is possible to set the type of the return argument with the `restype` attribute:

```
>>> hello.greet.restype = c_char_p   # C character pointer
>>> hello.greet()
'hello, world'
```

(might need to set `export LD_LIBRARY_PATH='<something>'`)

The use of Windows DLL is also possible with `ctypes`

. . . ctypes

ctypes example using c standard library:

```
>>> from ctypes import *
>>> c = CDLL('/lib/x86_64-linux-gnu/libc.so.6')
>>> c.strcmp('Finn', 'Finn')
0
>>> c.strcmp('Finn', 'Nielsen')
-8
```

CFFI

Common Foreign Function Interface for Python calling C code

Here an example to use the `strcmp` function in C from Python:

```
>>> from cffi import FFI
>>> ffi = FFI()
>>> ffi.cdef("int strcmp(const char *s1, const char *s2);")
>>> C = ffi.dlopen(None)
>>> C.strcmp('Finn', 'Finn')
0
```

Here `int strcmp(const char *s1, const char *s2);` is simply take from the man page of strcmp.

Low-level non-wrapped

See the documentation at

http://docs.python.org/2/extending/extending.html

See also a small example in (Langtangen, 2008, section 5.1.2). PDF available from DTU.

Cython

Write a Python file (possibly with extended Cython syntax for static types), compile to C and compile the C.

Cython is a fork of Pyrex.

Simplest example with compilation of a python file `helloworld.py`, containing `print("Hello, World")`:

```
$ cython --embed helloworld.py
$ gcc -I/usr/include/python2.7 -o helloworld helloworld.c -lpython2.7
$ ./helloworld
```

More: You can compile to a module instead (callable from Python); you can include static types in the Python code to make it faster (often these files have the extension *.pyx).

Other ways to make Python faster

PyPy: Just-in-Time compilation, see speed comparison.

numexpr: faster Numpy multi-threading computation

scipy.weave: Embedding of C in Python

Numba: Project from 2012. JIT via decorators in "ordinary" python and LLVM. Still pre-version 1.

Embedding Python in another language

A C-program (`embedding.c`):

```
#include <Python.h>

int main(int argc, char **argv)
{
    Py_Initialize();
    Py_Main(argc, argv);
    Py_Finalize();
}
```

Compile, link and execute and you get the Python prompt:

```
$ gcc -I/usr/include/python2.7 embedding.c -lpython2.7 -o embedding
$ ./embedding
```

. . . Embedding Python in another language

`PyRun_SimpleString`: Run a string as python code

`PyRun_SimpleFile`: Run a python file

and other commands. . .

More information

(Langtangen, 2008, Chapter 5) available through DTU: Combining Python with Fortran, C, and C++

Python Standard Library documentation for ctypes

SWIG versus Boost? Which one to choose? Some discussion here: https://dev.lsstcorp.org/trac/wiki/SwigVsBoostPython

Stefan Behnel, http://www.behnel.de/cython200910/talk.html, slides from talk by developer

Embedding Python in Another Application, official documentation.

William Nagel, Embedding Python in Your C Programs, Linux Journal, 2005.

. . . more information

IntegratingPythonWithOtherLanguages from the Python wiki.

Peter Toft et al., Kald af C/C++ kode fra Python, Version2.

Summary

You do not need to compile with `ctypes`. It gives you direct access to C-functions.

You can call functions in other language by writing wrappers.

Usually you would generate the wrappers automatically with, e.g., SWIG, Boost.Python or other.

References

Langtangen, H. P. (2008). *Python Scripting for Computational Science*, volume 3 of *Texts in Computational Science and Engineering*. Springer, Berlin, third edition edition. ISBN 978-3-642-09315-9.