Interfacing With Other Languages

Luís Pedro Coelho

Programming for Scientists

April 28, 2009

University of Pittsburgh

Carnegie Mellon

Python: Language & Implementation

Python is a language, but it's also a program.

Definition

A successful language is a language with more than one implementation.

CPython

The main Python implementation is CPython.

Other Implementations

- Stackless Python (C)
- Jython (Java)
- IronPython (.NET)
- PyPy (Python in Python)
- Pyjamas (Python to Javascript!)
- ...

Writing Extensions

Writing extensions to CPython can be done in C(++) or similar languages (Fortran).

Writing an Extension Example

```
def countchar(string,c):
    '''
    C = countchar(string,ch)

    Counts the number of instances of character ch
    in string string. Returns an integer.
    '''
    res = 0
    for ch in string:
        if c == ch:
            res += 1
    return res
```

Writing A Python Extension

```
PyObject* countchar(PyObject* self, PyObject* args) {
    const char* string;
    const char ch;
    if (!PyArg_ParseTuple(args,"sc",&string,&ch))
        return NULL;
    int res = 0;
    while (*string) {
        if (ch == *string) ++res;
        ++string;
    }
    return Py_BuildValue("i", res);
}
```

Writing a Python Extension

1. Get the arguments into C variables
2. Do computation
3. Put the arguments into Python variables
4. Return

Get the Arguments

PyArg_ParseTuple

Like printf (or scanf), takes a format string.

Powerful, but fragile.

Telling Python About Your Function

```
static PyMethodDef methods[] = {
    {"countchar",  countchar, METH_VARARGS,
            "countchar(str,ch)\n"
            "Counts the number of ..."},
    {NULL, NULL, 0, NULL} /* Sentinel */
};

PyMODINIT_FUNC initcountchar(void)
{
    (void) Py_InitModule("countchar", methods);
}
```

Embedding C++

Trivial.

Just remember to add a couple of extern ”C” here and there.

C++ Example

```
extern "C" {
    #include <Python.h>
}

...
const char* module_doc =
    "countchar module.\n\n"
    "This is a great module.\n";

extern "C"
void initcountchar()
  {
    (void)Py_InitModule3("countchar", methods, module_
  }
```

Structuring

Two Layers

1. Python: Massage the arguments
2. C(++): Do computation
3. Python: Massage results

Example

```
import _countchar

def countchar(s,ch):
    '''
    countchar(str,ch)

    Counts the ...
    '''
    if s is None:
        return 0
    return _countchar(str,ch)
```

Alternatives

SWIG

Does most of what you've seen automatically.

Boost.Python

Allows mixed language applications (Python/C++).

SWIG

Pros

- Easy to use.
- Widely used.
- Well supported by build tools.

Cons

- Error messages are not Pythonic.
- Leads to undocumented functions.
- C++ only: template support lacking.

Boost.Python

Pros

- Tight integration of C++/Python
- Amazing Technology

Cons

- Hard to use

Summary

1. Use swig as your first pass.
2. Write your own for control.
3. Use Boost.python for very large projects.

Fortran

There exists a tool, called f2py which is similar to swig for Fortran.

Scipy.weave: Inline C++ Code

```
import numpy as np
from scipy import weave
from scipy.weave import converters

p2 = np.zeros(N)
code = '''
for (int i = 0; i != N; ++i) {
    for (int j = 0; j != N; ++j) {
        p2(i) += p(i,j)*p(i,j);
    }
}
'''
weave.inline(
        code,
        ['N','p','p2'],
        type_converters=converters.blitz)
```

Scipy.Weave

Pros

- Very fast to use
- Convenient syntax

Cons

- Error messages can be hard to parse
- Your code will run differently if
  you distribute it to people without a C++ compiler.

Cython

Cython is language which extends Python to make it easier to write Python extensions.

Cython

Pros

- Well supported
- Familiar syntax

Cons

- Still in development (moving target)