Extending Python

Francisco Fernandez Castano

graphenedb.com

francisco.fernandez.castano@gmail.com @fcofdezc

November 21, 2015

https://github.com/fcofdez/talks/blob/master/extending/presentation_1.pdf

Overview

1. Motivations
2. Guidelines
3. Native extensions
4. CTypes
5. CFFI
6. Conclusions

Caution

- Huge topic
- *Toy* examples
- Unix ĈPython centric

Motivation

Why write in C?

Motivation

- **Speed**
- Using legacy code
- Integration

Motivation

Why is Python *slow*?

- Interpretation overhead
- Boxed arithmetic and automatic overflow handling
- Dynamic dispatch of operations
- Dynamic lookup of methods and attributes
- Everything can change on runtime
- Extreme introspective and reflective capabilities

Motivation

- Speed
- **Using legacy code**
- Integration

Motivation

- Speed
- Using legacy code
- **Integration**

Shared libraries

- Single copy in memory
- Runtime load

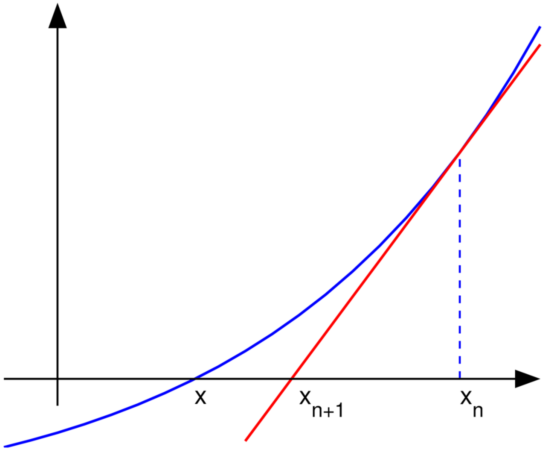
Native extensions

C API

Native extensions

Hello world, Newton method

Native extensions



Native extensions

```c
#include "Python.h"

static PyObject *
newton(PyObject *self, PyObject *args)
{
    float guess;
    float x;

    if (!PyArg_ParseTuple(args, "ff", &guess, &x))
        return NULL;

    while (fabs(powf(guess, 2) - x) > 0.01)
    {
        guess = ((x / guess) + guess) / 2;
    }

    return Py_BuildValue("f", guess);
}
```

Native extensions

```c
static PyMethodDef
module_functions[] = {
    {"newton", newton, METH_VARARGS, "Newton method."},
    {NULL}
};


PyMODINIT_FUNC
initnewton(void)
{
    Py_InitModule3("newton", module_functions, "Newton");
}
```

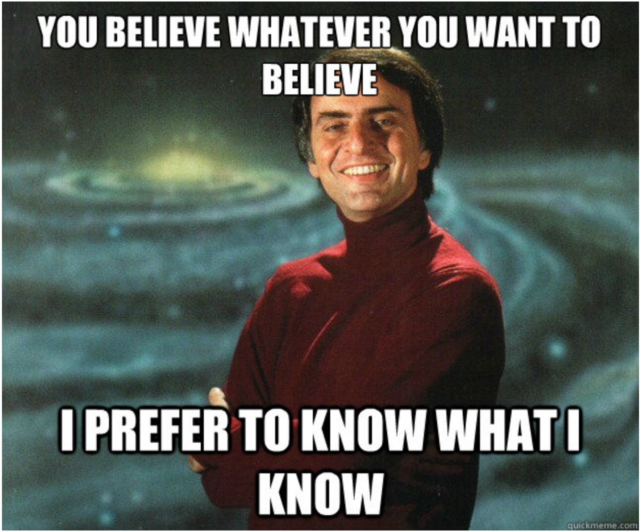
Native extensions

```python
from distutils.core import setup, Extension

setup(name='fosdem',
      version=1.0,
      ext_modules=[
        Extension('newton', ['newton.c'])])
```

Native extensions

```
Python 2.7.5 (default, Nov  3 2014, 14:26:24)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-7)] on linux2
>>> import newton
>>> newton.newton(1, 2)
1.4166667461395264
>>>
```

How?


YOU BELIEVE WHATEVER YOU WANT TO BELIEVE
I PREFER TO KNOW WHAT I KNOW
quickmeme.com

Native extensions

```
NAME
dlopen--load and link a dynamic library or bundle

SYNOPSIS
#include <dlfcn.h>

void*
dlopen(const char* path, int mode);
```

Native extensions

```
$ nm -g newton.so
00000000002010a0 B __bss_start
                 w __cxa_finalize@@GLIBC_2.2.5
00000000002010a0 D _edata
00000000002010a8 B _end
0000000000000944 T _fini
                 w __gmon_start__
00000000000006d0 T _init
0000000000000920 T initnewton
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
                 w _Jv_RegisterClasses
                 U PyArg_ParseTuple
                 U Py_BuildValue
                 U Py_InitModule4_64
```

Native extensions

cpython/Python/dynload_shlib.c

```
handle = dlopen(pathname, dlopenflags);

if (handle == NULL) {
    const char *error = dlerror();
    if (error == NULL)
        error = "unknown dlopen() error";
    PyErr_SetString(PyExc_ImportError, error);
    return NULL;
}
if (fp != NULL && nhandles < 128)
    handles[nhandles++].handle = handle;
p = (dl_funcptr) dlsym(handle, funcname);

return p;
```

Native extensions

Memory management

- Manual memory management
- Python GC - RC
- Cycle detector
- Py_INCREF(x) Py_DECREF(x)

Native extensions

Error management

- Return NULL as a convention
- Register exceptions

Native extensions

Error management

```c
if (err < 0) {
  PyErr_SetString(PyExc_Exception, "Err");
  return NULL;
}
```

Native extensions

Python 3 differences

```c
static struct PyModuleDef examplemodule = {
  PyModuleDef_HEAD_INIT,
  "newton",
  "newton module doc string",
  -1,
  module_functions,
  NULL,
  NULL,
  NULL,
  NULL};
```

Native extensions

Python 3 differences

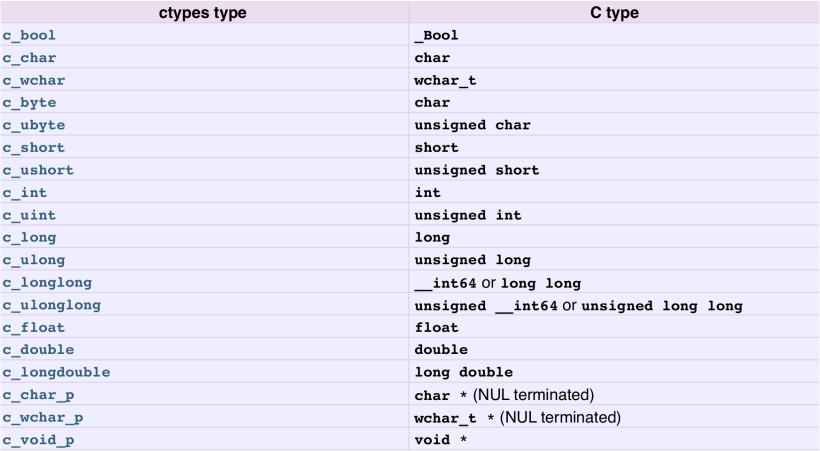
```
PyMODINIT_FUNC
PyInit_sum(void)
{
PyModule_Create(&examplemodule);
}
```

CTypes

- Advanced FFI for Python
- Allows call functions from Shared libs
- Create, access, manipulate C data types

CTypes

Types correspondence



ctypes type	C type
`c_bool`	`_Bool`
`c_char`	`char`
`c_wchar`	`wchar_t`
`c_byte`	`char`
`c_ubyte`	`unsigned char`
`c_short`	`short`
`c_ushort`	`unsigned short`
`c_int`	`int`
`c_uint`	`unsigned int`
`c_long`	`long`
`c_ulong`	`unsigned long`
`c_longlong`	`__int64` or `long long`
`c_ulonglong`	`unsigned __int64` or `unsigned long long`
`c_float`	`float`
`c_double`	`double`
`c_longdouble`	`long double`
`c_char_p`	`char *` (NUL terminated)
`c_wchar_p`	`wchar_t *` (NUL terminated)
`c_void_p`	`void *`

CTypes

Structs

```python
from ctypes import *

class POINT(Structure):
    _fields_ = [("x", c_int), ("y", c_int)]

class RECT(Structure):
    _fields_ = [("upperleft", POINT),
                ("lowerright", POINT)]
```

CTypes

Example

- Implemented fibonacci as c function
- Map as Python code
- Measure differences between Python and C

CTypes

```c
int fib(int n){
  if (n < 2)
      return n;
  else
      return fib(n - 1) + fib(n - 2);
}
```

CTypes

```python
import ctypes
lib_fib = ctypes.CDLL("libfib.so")

def ctypes_fib(n):
    return lib_fib.fib(ctypes.c_int(n))

def py_fib(n):
    if n < 2:
        return n
    else:
        return py_fib(n-1) + py_fib(n-2)
```

CTypes

```
In [3]: %timeit fib.ctypes_fib(20)
10000 loops, best of 3: 63.8 micro s per loop

In [4]: %timeit fib.py_fib(20)
100 loops, best of 3: 3.62 ms per loop
```

CTypes

Fortran example

- Use of existing fortran code
- Take random code at github
- https://github.com/astrofrog/fortranlib
- Wrap using ctypes

CTypes

Fortran example

```
real(dp) function mean_dp(x, mask)
    implicit none
    real(dp),intent(in) :: x(:)
    logical,intent(in),optional :: mask(:)
    if(present(mask)) then
       mean_dp = sum(x, mask=mask)/size(x)
    else
       mean_dp = sum(x)/size(x)
    end if
end function mean_dp
```

CTypes

Fortran example

```
gfortran -fPIC -shared statistic.f90 -o lib_statistics.so
```

CTypes

Fortran example

```
bin git:(master) -> nm -g lib_statistics.so
001eab T ___lib_statistics_MOD_clipped_mean_dp
000afc T ___lib_statistics_MOD_clipped_mean_sp
00306c T ___lib_statistics_MOD_mean_dp
001c55 T ___lib_statistics_MOD_mean_sp
002db0 T ___lib_statistics_MOD_median_dp
0019b0 T ___lib_statistics_MOD_median_sp
002544 T ___lib_statistics_MOD_quantile_dp
00115a T ___lib_statistics_MOD_quantile_sp
002299 T ___lib_statistics_MOD_variance_dp
000ec3 T ___lib_statistics_MOD_variance_sp
U __gfortran_arandom_r4
U __gfortran_arandom_r8
U __gfortran_os_error
U __gfortran_pack
U __gfortran_pow_i4_i4
U __gfortran_runtime_error
U __gfortran_runtime_error_at
U __gfortran_st_write
```

CTypes

Fortran example

```
from ctypes import *

# Statistics fortran lib
st_lib = CDLL(’lib_statistics.so’)

mean = st_lib.__lib_statistics_MOD_mean_dp
mean.argtypes = [POINTER(c_float*2)]
mean.restype = c_float

vals = (c_float*2)(2.0, 3.0)

print mean(vals)
```

CTypes

CTypes source

cpython/Modules/_ctypes/callproc.c

```c
static PyObject *py_dl_open(PyObject *self, PyObject *args)
{
  char *name;
  void * handle;
  #ifdef RTLD_LOCAL
    int mode = RTLD_NOW | RTLD_LOCAL;
  #else
    /* cygwin doesn't define RTLD_LOCAL */
    int mode = RTLD_NOW;
  #endif
  if (!PyArg_ParseTuple(args, "z|i:dlopen", &name, &mode))
     return NULL;
  mode |= RTLD_NOW;
  handle = ctypes_dlopen(name, mode);
  .
  return PyLong_FromVoidPtr(handle);
}
```

CFFI

- Advanced FFI for Python
- Allows call functions from Shared libs
- Create, access, manipulate C data types
- Both API and ABI access

CFFI

Mostly the same as CTypes

CFFI

Recommended way to extend PyPy

CFFI

ABI

```python
from cffi import FFI

ffi = FFI()
ffi.cdef("""int printf(const char *format, ...);""")
C = ffi.dlopen(None)
arg = ffi.new("char[]", "world")
C.printf("hi there, %s!\n", arg)
```

CFFI

ABI- Fibonacci

```python
from cffi import FFI

ffi = FFI()
ffi.cdef("""int fib(int n);""")
libfib = ffi.dlopen(’libfib.so’)
libfib.fib(10)
```

CFFI

API Level

```
import cffi
ffi = cffi.FFI()

ffi.cdef("""int fib(int n);""")

ffi.set_source("_fib", r’’’
int fib(int n){
if ( n < 2 )
return n;
else
return fib(n-1) + fib(n-2);
}’’’)

if __name__ == ’__main__’:
  ffi.compile()
```

CFFI

API Level

```python
from _fib import fib
print(fib(10))
```

CFFI

API Level

```python
from _fib import fib
print(fib("asdasd"))
```

CFFI

API Level

```
Traceback (most recent call last):
File "fib.py", line 16, in <module>
print lib.fib("asd")
TypeError: an integer is required
```

CFFI

API Level

```python
from cffi import FFI
ffi = FFI()
ffi.cdef("""typedef struct { float x, y; } point;""")
point = ffi.new("point *")
point.x = 2.0
point.y = 3.0
```

CFFI

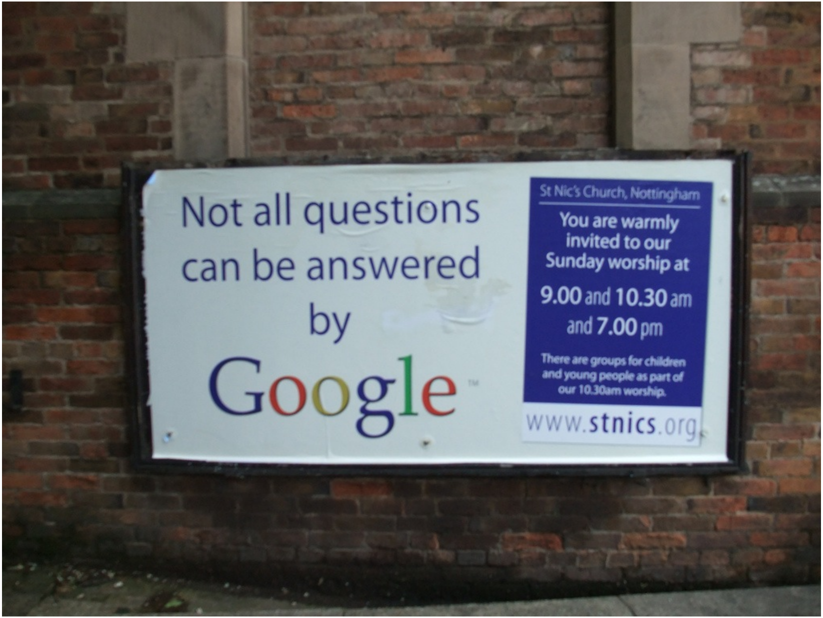
Internals

cffi/c/_cffi_backend.c

```c
static PyObject *
b_load_library(PyObject *self, PyObject *args)
{
    void *handle;
    DynLibObject *dlobj;

    if ((flags & (RTLD_NOW | RTLD_LAZY)) == 0)
        flags |= RTLD_NOW;

    printable_filename = filename_or_null ? filename_or_null :
    handle = dlopen(filename_or_null, flags);

    dlobj = PyObject_New(DynLibObject, &dl_type);
    dlobj->dl_handle = handle;
    dlobj->dl_name = strdup(printable_filename);
    return (PyObject *)dlobj;
}
```

Conclusions

- Three different ways
- Same principles
- Less portable - More portable
- Harder - Easier



Not all questions can be answered by Google™

St Nic's Church, Nottingham

You are warmly invited to our Sunday worship at

9.00 and 10.30 am and 7.00 pm

There are groups for children and young people as part of our 10.30am worship.

www.stnics.org