Tema 7 Bibliotecas externas

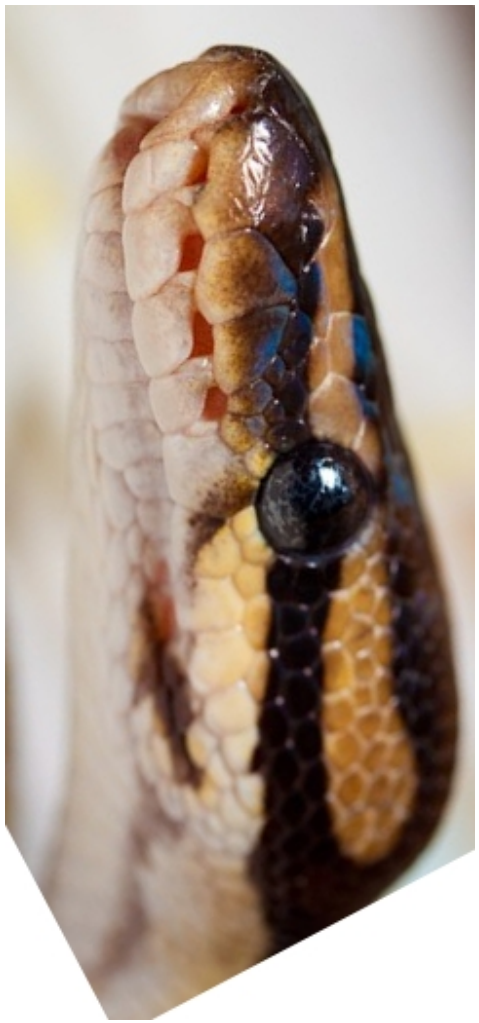
Curso de Python Avanzado

Juan Pedro Bolívar Puente

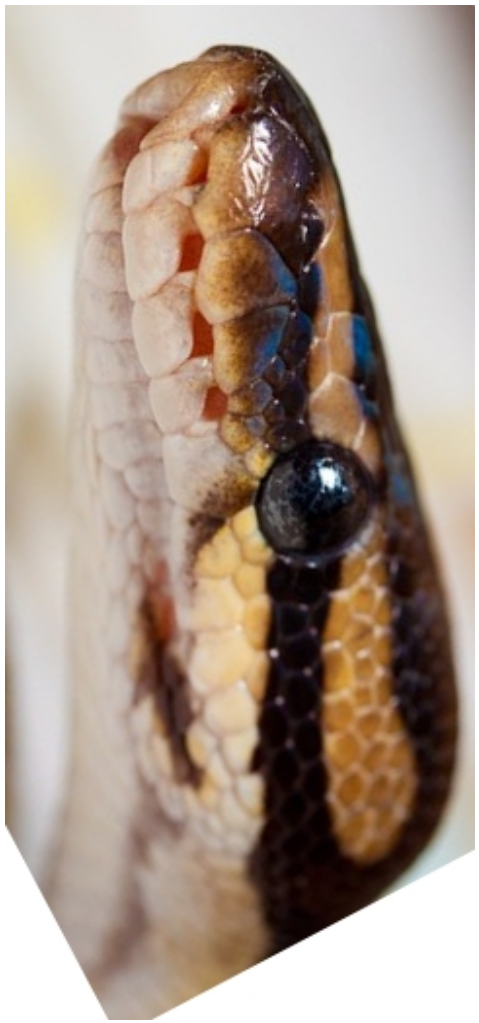
Instituto de Astrofísica de Andalucía

Mayo de 2011

Índice



Índice



Introducción

¡Python no es siempre suficiente!

Motivos ...

- Rendimiento
- Paralelismo
- Reutilización
- Código *legacy*

Extensiones de Python

Una extensión es un módulo escrito en C

Problemas ...

- No suele funcionar fuera de cpython
- Requiere mucho código

Ejemplos de la cabecera <Python.h>

- `PyObject* obj`
- `PyArg_ParseTuple`
- `Py_INCREF, Py_DECREF`
- `Py_InitModule, PyModule_AddObject`

Ejemplo de extensión en Python

```
#include <Python.h>

static PyObject*
mymod_func(PyObject *self, PyObject *args)
{
    PyObject *a, *b;
    if (!PyArg_UnpackTuple(args,"func",2,2,&a,&b))
        return NULL;
    return PyNumber_Add (a, b);
}

static PyMethodDef mymod_methods[] = {
    {"func", some_func, METH_VARARGS, "Una␣funcion"},
    {NULL, NULL}
};
```

Ejemplo de extensión en Python

```
PyMODINIT_FUNC init_mymod(void)
{
    Py_InitModule3 ("mymod",
                    mymod_methods,
                    "Some module");
}
```

SWIG

Genera todo el código redundante desde una descripción de alto nivel

Ventajas ...

- Abstrae la generación de código
- Usado en proyectos serios
- También sirve para PHP, Perl, Ruby, Lisp, Java...

SWIG

Genera todo el código redundante desde una descripción de alto nivel

Desventajas ...

- Requiere aprender su lenguaje
- Código generado por ordenador...
  - Difícil de leer
  - Infernal para depurar
- Añade pasos y dependencias

Ejemplo de módulo en SWIG

Interfaz example.i

```
%module example
%{
extern double My_variable;
extern int fact(int n);
extern char *get_time();
%}

extern double My_variable;
extern int fact(int n);
extern char *get_time();
```

Ejemplo de módulo en SWIG

A menudo basta...

```
%module example
%{
#include "header.h"
%}
%include "header.h"
```

Y para generar...

```
$ swig -python example.i
$ gcc -c example.c example_wrap.c \
      -I/usr/local/include/python2.1
```

... bienvenido a ctypes

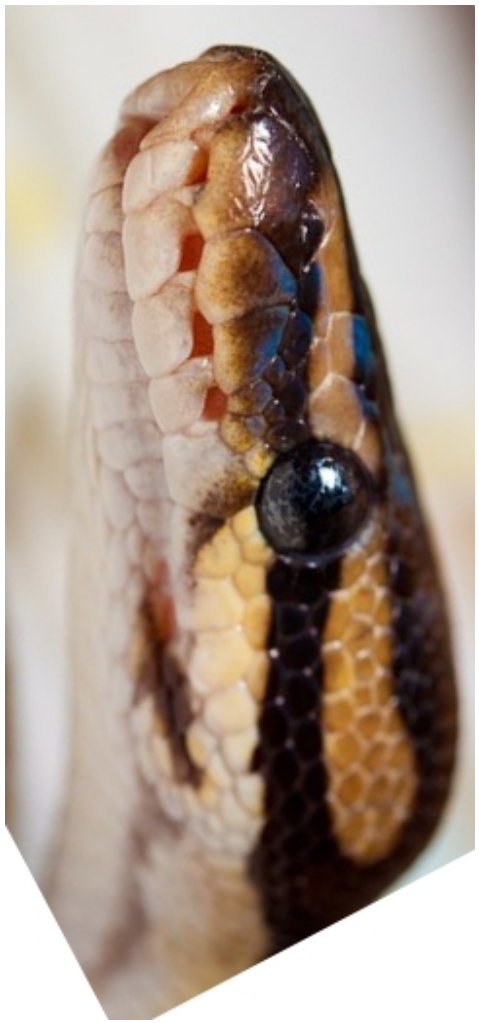
Hacemos la interfaz ¡desde Python!

Ventajas...

- Podemos usar directamente cualquier .dll, .so, .dylib
- No hay que salir de Python
- Incluído en la biblioteca estándar
- Interacciona bien con NumPy!

Python $\geq$ 2.5

Índice



Librerias en ctypes

Una librerías es un objeto Python

`ctypes.CDLL (name, mode, handle)`

Librerías compartidas con estilo C

Por ejemplo Fortran también usa esta esta convención

`ctypes.PyDLL (name, mode, handle)`

Librerías con extensiones de Python

El GIL no se libera ...

Otros... En Windows... WINDLL, OLEDLL

Paréntesis

Compilar bibliotecas suele requerir parámetros especiales, en GNU/Linux esto es ...

Compilar objeto de biblioteca compartida

```
$ gcc -fPIC -c -o fichero.o fichero.c
```

Enlazar biblioteca compartida

```
$ ld -shared -ldl -o fichero.so fichero.o ...
```

Ver *script* cclib en el código adjunto...

El cargador de bibliotecas

Mejor usar ctypes.LibraryLoader

Accesible desde `ctypes.[cdll|pydll]`

- Provee `LoadLibrary`
- Sobrecarga el operador `[ ]`
- Sobrecarga `gettattr`
- ¡Cachea resultados!

Ejemplillo

```
import ctypes

_cstd = ctypes.cdll ['libc.so.6']
_cmath = getattr (ctypes.cdll,
                  'libm.so.6')

_cstd.printf ("Hola␣mundo!")
```

Las funciones son atributos del objeto biblioteca

Buscando bibliotecas

`ctypes.util.find_library`
busca una biblioteca

- En Linux usa `gcc, ldconfig y objdump`
- En Mac simplemente busca en los "sitios típicos"
- En Windows busca en sitios típicos, pero no tiene esquema de nombres y casca mucho

En nuestras propias bibliotecas
hardcodead el nombre en tiempo de instalación

Ejemplillo

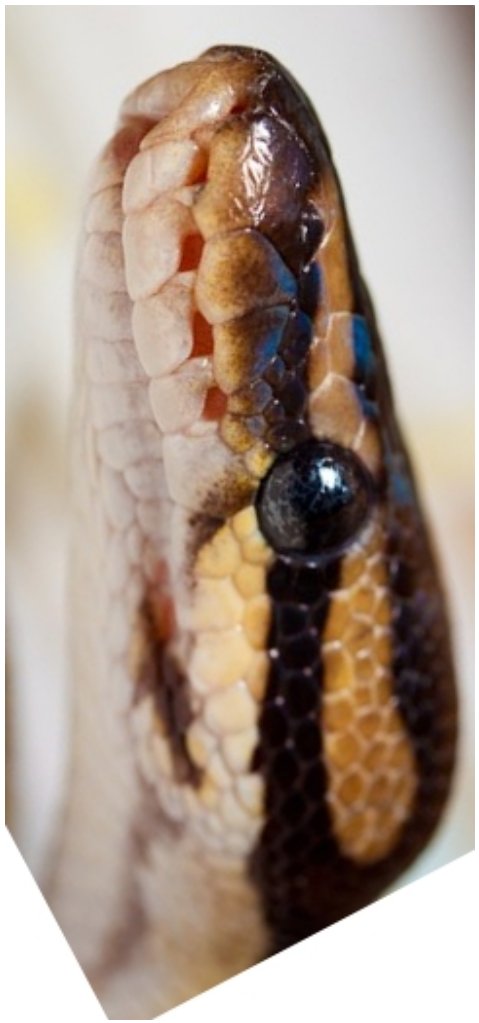
```
import ctypes
from ctypes.util import find_library

_cstd = ctypes.cdll[find_library('c')]

_cmath = getattr (ctypes.cdll,
                  find_library ('m'))

_cstd.printf ("Hola␣mundo!")
```

Índice



Funciones ...

Las funciones son atributos del objeto CDLL/PyDLL

- Tienen tipo `ctypes._FuncPtr`
- Son structure*callables* normales
- Hay que convertir los parámetros

Conversión de tipos

- `str`, `unicode`, `int`, `long`, `NoneType`
  se convierten automáticamente

  El ejemplo con `printf()`
- En otro caso, especificar la signatura de la función
  `func.argtypes = sequencia-de-ctypes...`
  `func.restype = ctype..`

Tipo Python $\Rightarrow$ ctype $\Rightarrow$ tipo C

Ejemplo ...

```
from ctypes import *
from ctyes.util import *

_cmath = cdll [find_library ('m')]
_fabs = _cmath.fabs
print _fabs (c_double (5.5))

_fabs.argtypes = (c_double,)
_fabs.restype  = c_double
print _fabs (5.5)
```

Comprobando errores

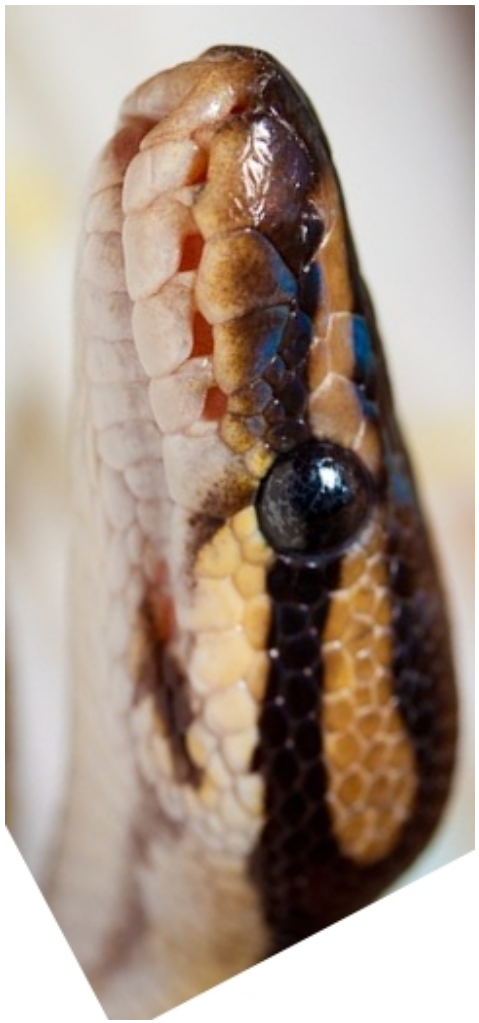
En errcheck podemos instalar un comprobador de errores

Puede servir para convertir errores en excepciones

```
def io_errcheck (res, func, args):
    if not res:
        raise IOError, 'Error opening file'
    return res
_cstd.fopen.errcheck = io_errcheck

_cstd.fopen ('notexist', 'r')
```

Índice



Tipos básicos en ctypes

Mutable

- `c_char`, `c_wchar`, `c_byte`, `c_ubyte`
- `c_short`, `c_ushort`
- `c_int`, `c_uint`, `c_long`, `c_ulong`
- `c_float`, `c_double`

Inmutable

- `c_char_p`, `c_wchar_p`, `c_void_p`
- `create_string_buffer` para la mutabilidad

Al valor de una instancia se accede mediante `value`

Ejemplo ...

```
from ctypes import *
from ctypes import util
_cstd = cdll [find_library ('c')]

s = 'Hola␣mundo!'
_cstr.printf ("%s\n", c_char_p (s)) # Ok
_cstd.scanf ("%s", c_char_p (s))  # Mal!!
print s == 'Hola␣mundo!' # OMG!

s = create_string_buffer (128)
_cstd.scanf ("%s", s)  # Mejor
print s.value
```

Punteros...

- Construimos un tipo puntero con `POINTER`
- Construimos un puntero con `pointer`

  $\Rightarrow$ dereferenciamos con `contents` y `operator[]`

```
i  = c_int (5)
pi = pointer (i)
i.contents = 10
print i
assert type (pi) == POINTER (c_int)
```

Punteros...

Si no necesitamos el puntero en Python, `byref` es más eficiente que `pointer`

Suele usarse para el *paso por referencia*

```
i = c_int ()
_cstd.scanf ("%i", byref (i))
```

Arreglos ...

`tipo * N` da un tipo arreglo sobre elementos de *tipo*

```
int3 = c_int * 3

a3 = int3 ()
print list (a3)

a3 = int3 (* range (3))
print list (a3)
```

struct y union

Se definen heredando de `Structure` o de `Union`

`_fields_` contiene una lista de `(nombre, tipo [, bitfield])` con los campos

`_fields_` puede definirse tras la clase para las estructuras recursivas

Ejemplo ...

```
class c_tm (Structure):
    _fields_ = (
        ('tm_sec',    c_int),
        ('tm_min',    c_int),
        ('tm_hour',   c_int),
        ('tm_day',    c_int),
        ('tm_mon',    c_int),
        ('tm_year',   c_int),
        ('tm_wday',   c_int),
        ('tm_yday',   c_int),
        ('tm_isdst',  c_int))
```

Ejemplo ...

```
_cstd.localtime.restype = \
    POINTER (c_tm)

t  = c_int (_cstd.time (0))
lt = _cstd.localtime (byref (t))
lt = lt.contents

print "%d/%02d/%02d" % (
          lt.tm_year + 1900,
          lt.tm_mon,
          lt.tm_day)
```

Punteros a funciones ...

Definimos un tipo a puntero a función con:

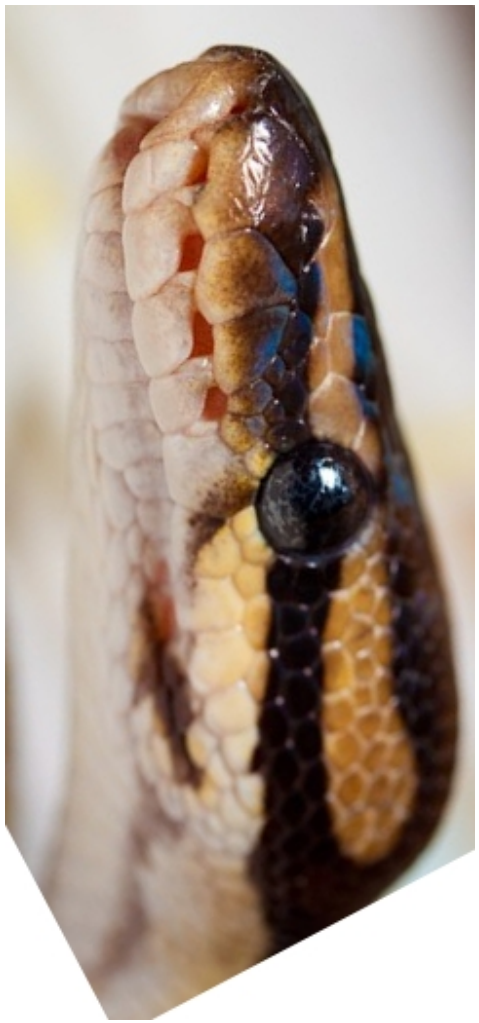
`CFUNCTYPE (rettype [, argtypes ...])`

```
cmpfunc = CFUNCTYPE (
    c_int, POINTER (c_int), POINTER (c_int))

@cmpfunc
def mycmp (a, b): return cmp (a[0], b[0])

arr = (c_int * 10) (* reversed (xrange (10)))
_cstd.qsort (arr, 10, sizeof (c_int), mycmp)
print list (arr)
```

Índice



Compatibilidad con Numpy

Los arreglos de Numpy exponen sus entrañas en el atributo `ctypes`

Esto a su vez contiene...

- `data`, contenido como `c_void_p`
- `data_as (ptrtype)`, contenido como `ptrtype`
- `shape`, el tamaño como `c_long * ndim`
- `stride`, el *stride* como `c_long * ndim`

Añadir dibujito!

Comprobando tipos

Podemos imponer restricciones sobre los punteros con

```
numpy.ctypeslib.ndpointer (
[ dtype, ndim, shape, flags ])
```

Además, se encarga de realizar la conversión

flags es una cadena o lista de cadenas:

- C_CONTIGUOUS / C / CONTIGUOUS
- F_CONTIGUOUS / F / FORTRAN
- OWNDATA / O
- WRITEABLE / W
- ALIGNED / A
- UPDATEIFCOPY / U

Ejemplo ...

Biblioteca en C

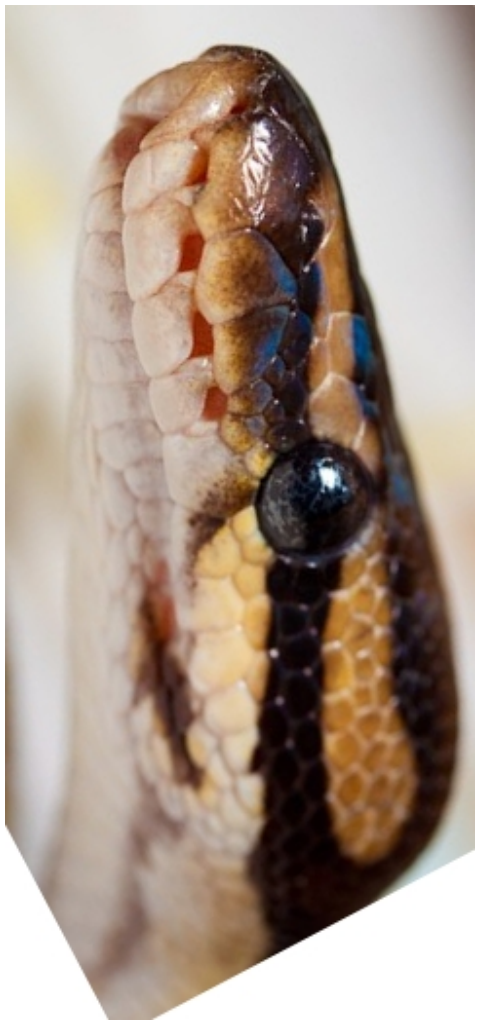
```
float dotprod3 (float* a, float* b)
{
  return a[0] * b[0] +
         a[1] * b[1] +
         a[2] * b[2];
}
```

```
from numpy.ctypeslib import ndpointer
from numpy import array
_funcs = CDLL ('./funcs.so')

vec3 = ndpointer (dtype = 'float32',
                  ndim  = 1,
                  shape = (3,),
                  flags = 'C_CONTIGUOUS')
_funcs.dotprod3.argtypes = vec3, vec3
_funcs.dotprod3.restype  = c_float

print _funcs.dotprod3 (
    array ([1., 1., 1.], dtype = 'float32'),
    array ([0., 0., 3.], dtype = 'float32'))
```

Índice



Metodología

¡Haced envoltorios pitónicos!

Por cada biblioteca en C, escribir un módulo en Python

- El objeto biblioteca es una variable privada global
- Envolver tipos complejos de C en clases de Python
- Simplificar la interfaz con métodos y funciones

Recursos adicionales

¿Preguntas?

Muchas gracias por su atención.