Scripting C with Python

Duncan Booth
duncan.booth@suttoncourtenay.org.uk

Introduction

- Why?
- What do I need to know?
- Examples
- Error handling
- Memory allocation
- Multi-threading

Why mix C and Python?

- Add scripting to your C/C++ application (embedding)
- Glue
  - use Python to write different front-end drivers to a common library
- Use a C library
  - for speed
  - because it implements functionality not present in Python
- Test Driven Development
  - test C code using Python

When to mix them

- Write the Python first
  - Writing Python is faster.
  - Use it to work out the best implementation, then convert speed sensitive code to C
- then write C
  - (but only if you have to)

Real life intervenes

- More usually
  - Add Python to an existing codebase
  - Existing design decisions may make it hard

Embedding vs Extending

- Embedding
  - Add scripting to an existing program.
  - Main program is C/C++
  - Python interpreter is called when needed
  - Almost always used with **Extending**
- **Extending**
  - Gives Python access to a module written in C/C++
  - add functionality to Python
  - speed up where Python is slow
    - (although Psyco is an alternative)

C API or using a toolkit

- Is there a pre-existing library that does the job?
  - Don't reinvent the wheel
- Do you just want to call C functions in a dll/so
  - look at *ctypes*
- Do you prefer writing Python rather than C?
  - yes: Look at *Pyrex*
  - no: still look at *Pyrex*

C API or using a toolkit

- Do you need support for other languages (Perl, Tcl etc.)
  - *SWIG*
- Are we talking C++ here?
  - *Boost.Python*

ctypes

- Call C functions exported from library
- Create and manipulate C compatible types and data structures
- No need to write any C code
- No C compiler required

ctypes Example

Pyrex

- Python syntax with extensions
  - define Python functions, classes
    - usable from Python and Pyrex
  - define classes implemented in C
    - usable from Python and Pyrex
  - define C functions
    - callable only from Pyrex and C
- Supports embedding as well as extending
- Lets you use Python syntax running at C speed

Pyrex Example

Handling errors and exceptions

- C error returns must be converted to Python exceptions
- C++ exceptions must be caught and converted to Python exceptions
  - may require 'C' wrapper around 'C++' code
- Calling Python API
  - errors indicated by return code
  - Python exception will have been set
  - must propagate error return upwards

Handling errors and exceptions

- ctypes
  - checks stack to detect wrong number of arguments
  - catches memory faults
- Pyrex
  - Automatic checking for Python API errors

Memory Allocation

- Objects used in Python should be allocated through Python API
  - Lifetime determined by
    - reference counting
    - garbage collection
- C applications use custom memory schemes
  - object lifetime may not match Python's expectations
  - copy or wrap objects?
- Callbacks into Python
  - avoiding duplicate wrapped objects

Multi-threaded applications

- Global Interpreter Lock
  - Release GIL before calling long running code
  - Reclaim GIL on return
- Callbacks
  - Claim/release GIL on callbacks
  - Need thread data for this
- When GIL is released
  - Python objects may mutate
  - Python objects may be freed
  - Don't borrow references

Summary

- Python & C work great together
- Python
  - Fast development
- C
  - Runs faster
  - Existing libraries