Extending Python with Rust

Introduction and a hands-on demo of writing Python extension in Rust

This talk

- Why?
- Why Rust?
- Binginds
- Dynamic library
- Web service example
- Docker example
- Compile and distribute (PIP example)

Why?

- Speed. The dynamic nature of Python means it is notoriously slow at some things, and not great on parallel calculations.
- Reusability. Why port code when you can use it directly?
- Cooperation. If one team writes Python and another - Rust, and they need to use each other's code.
- Migration. If you want to re-write your entire Python codebase to Rust, you could do that module by module.

How?

- C extension
- Cython - magic!
- Numba - JIT-compilation

Why Rust?

- Rust is an innovative compiled language with an accent on safety.
- Rust is one of the most loved languages by developers.
- ...and companies too
- Minimal runtime (e.g. no garbage collector)

Yet Rust has a somewhat steep learning curve. In many cases, one may want to split their codebase, and write some critical code in Rust, and some more common code in Python.

Bindings

- Rust-cpython (which we're going to use for the examples)
- PyO3 - fork of rust-python, only on nightly Rust
- Python as a scripting language in Rust programs (not in this talk)
- Cargo - Rust package manager and CLI
- The Rust book and the Cargo book

Dynamic library

- Simple Rust code - just two files
- Will be building a dynamic library - only need to set crate-type
- .dll file on Windows, .so on Linux, or .dylib on Mac
- Rust-cpython provides wrappers for Python
- Building for Mac requires additional linker arguments
- Rename to mylib.so
- DEMO

DEMO

Web service

- More "real-world" example
- A Python web-service that calls Rust library and returns the result to user
- Will use Flask + the library we just built
- Can run it with “FLASK_APP=main.py flask run” but for production, we would like more: Continuous Integration/Deployment
- Will build a Docker image

Docker

- Will use Gunicorn as a web-server
- Will use a multi-stage build
- The resulting Docker image will not contain any Rust artifacts - only a compiled binary
- Will build for Linux this time
- DEMO

DEMO

Compile and distribute

- What if we want to keep Rust and Python code separately?
- A Rust team and a Python team
- Corporate PIP repository
- “pip install mylib”
- Setuptools-rust
- Building from source vs wheels
- Can remove the first, Rust stage from our Dockerfile
- DEMO

DEMO

Thanks!

https://github.com/moor84