Extending Python with Rust: a hands-on introduction to PyO3

Ivan Carvalho

Agenda

- Introduce the tools
- Build an extension using PyO3
- Understand what problems Python extensions in Rust solve

About me

- Maintainer of `rustworkx` (peaked at the Top 1% of PyPI packages)
- User of PyO3 since 2021
- Casually contributed features I needed to PyO3 upstream

Why this presentation

Rust has taken over the Python ecosystem!

- Popular libraries like `pydantic` and `cryptography` use Rust
- Python tooling also uses Rust like:
  - Astral's uv
  - Microsoft's Python Environment Tools
  - Facebook's Pyrefly

But why? Hopefully you'll understand by the end.

Tools we are going to be using

We don't assume you'll be familiar with all of them. Rust users will know Rust tools, Python users will known Python tools.

- PyO3
- Maturin
- pip
- PyPI
- Cargo
- Crates.io

Motivating Problem

To give us a concrete goal, this presentation will build an extension that can decode JPEG XL images.

As of 2025, iPhones can now take photos and save in the JPEG XL file format. It's feasible you'd find this file format in the wild!

Maturin

Maturin is a Python build tool provided by the PyO3 developers. It helps building Rust code as extensions.

Maturin can be installed with `pip install maturin`.

The initial draft of the repository was the output of `maturin new`.

`maturin develop` installs the extension locally for development.

`maturin build` is used to package a wheel.

Manifest files

These are the two files from `maturin new`, edited by me.

Cargo.toml

```toml
[package]
name = "jxl_demo"
version = "0.1.0"
edition = "2021"

[lib]
name = "jxl_demo"
crate-type = ["cdylib"]

[dependencies]
ndarray = "0.16"
pyo3 = {
    version = "0.26.0",
    features = [
        "abi3",
        "extension-module"
    ]
}
numpy = "0.26"
jxl-oxide = "0.11.4"
```

pyproject.toml

```toml
[build-system]
requires = ["maturin>=1.9,<2.0"]
build-backend = "maturin"

[project]
name = "jxl_demo"
requires-python = ">=3.10"
classifiers = [
    # ommited
]
dynamic = ["version"]
dependencies = [
    "pillow>=10.0",
    "numpy>=2.0",
]

[tool.maturin]
features = ["pyo3/extension-module"]
```

Dependencies

This demo is only possible thanks to `jxl-oxide` and `pyo3` being easily available on crates.io.

Rust arguably has more friendly dependency management than Python. We'll not discuss the Python packaging ecosystem.

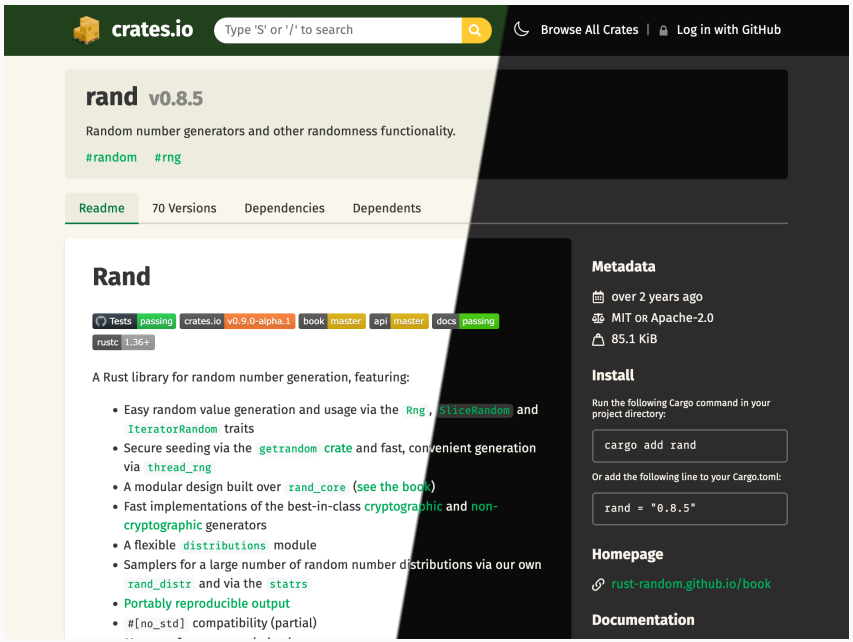

Figure 1: crates.io

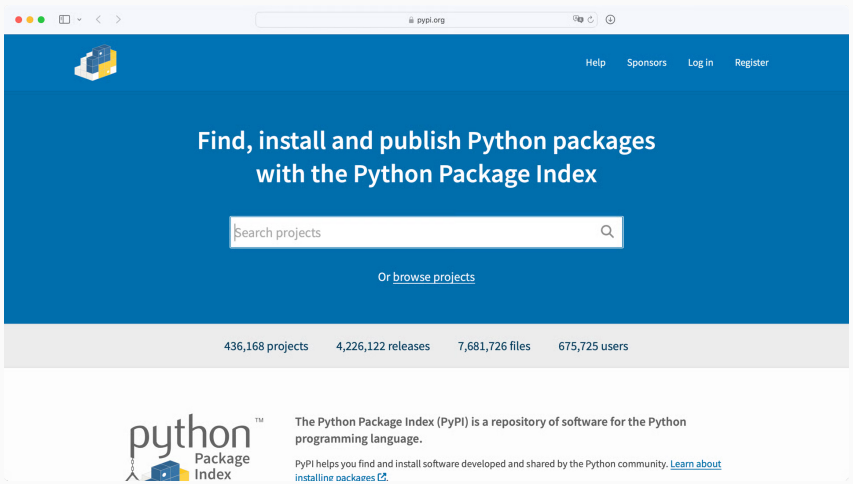

Figure 2: PyPI

PyO3

The glue between Python and Rust.

PyO3 exposes:

- Python types to Rust e.g. `PyAny`
- Conversion between Rust and Python types
- Rust functions to Python via `#[pyfunction]`
- Rust strucs to Python via `#[pyclass]`

Python	Rust	Rust (Python-native)
`object`	-	`&PyAny`
`str`	`String`, `Cow<str>`, `&str`	`&PyUnicode`
`bytes`	`Vec<u8>`, `&[u8]`	`&PyBytes`
`bool`	`bool`	`&PyBool`
`int`	Any integer type (`i32`, `u32`, `usize`, etc)	`&PyLong`
`float`	`f32`, `f64`	`&PyFloat`
`complex`	`num_complex::Complex`[1]	`&PyComplex`
`list[T]`	`Vec<T>`	`&PyList`
`dict[K, V]`	`HashMap<K, V>`, `BTreeMap<K, V>`	`&PyDict`
`tuple[T, U]`	`(T, U)`, `Vec<T>`	`&PyTuple`

Figure 3: Incomplete conversion list for PyO3

PyO3 In Practice