Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

High-Performance Python with Zig

Class Duration

7 hours of live training delivered over 1 day.

Student Prerequisites

- All students should have significant experience with Python
- Working knowledge of Zig: pointers, slices, error unions, and allocators
- The class has no review of the Python or Zig programming languages
- If you need to learn Zig, please consider the Zig Essentials course first

Target Audience

This course is for Python developers who need to accelerate CPU-bound Python code and want Zig as the native layer. It suits teams who value Zig's small toolchain, C-ABI fluency, and effortless cross-compilation, and engineers comparing Zig against the approaches covered in our High-Performance Python with C and C++, High-Performance Python with Cython, and High-Performance Python with Rust courses before committing to one.

Description

This one-day course is for Python developers who want to rewrite CPU-bound hot paths in Zig and ship the result as a normal Python package. Because Zig speaks the C ABI natively and imports `Python.h` directly with `@cImport`, a Zig source file can be a CPython extension module with no binding generator at all — the course starts there, teaching `PyObject*` handling, argument parsing, and reference counting discipline from Zig. It then moves up a level to Ziggy Pydust, the actively maintained toolkit for building Python extensions in Zig, which uses comptime to wrap and unwrap arguments between native Zig types and Python objects and ships a pytest plugin for running Zig tests from your Python test suite. The course is candid about ecosystem maturity: Pydust targets CPython 3.11+ and historically trails new Zig compiler releases by a few months, so teams learn both the high-level and raw C API routes and when each is the right call.

The distribution story is where Zig shines: the same `zig` binary that compiles your extension is a zero-dependency C cross-compiler, so building manylinux, musllinux, macOS, and Windows wheels with cibuildwheel is

dramatically simpler than with a traditional toolchain zoo. Students benchmark their Zig extensions honestly against pure Python, Cython, Rust, and C++ equivalents — cross-referencing the sibling courses High-Performance Python with Cython, High-Performance Python with Rust, and High-Performance Python with C and C++ — examine buffer protocol and NumPy interop for zero-copy data exchange, consider what officially supported free-threaded Python (PEP 703 / PEP 779) means for a Zig extension author, and finish by packaging a wheel for PyPI.

Learning Outcomes

- Profile Python applications to identify the hot paths actually worth rewriting in Zig
- Write a CPython extension module in pure Zig via the C ABI and `@cImport`, handling `PyObject*`, argument parsing, and reference counting correctly
- Build extensions with Ziggy Pydust and evaluate its comptime argument wrapping, pytest integration, and version-support trade-offs
- Exchange data with NumPy and other libraries zero-copy through the buffer protocol
- Cross-compile wheels for Linux (manylinux and musllinux), macOS, and Windows with `zig cc` and cibuildwheel
- Benchmark Zig extensions honestly against Cython, Rust, and C/C++ alternatives and present defensible numbers
- Reason about the GIL, releasing it for CPU-bound Zig work, and the implications of free-threaded Python (PEP 703 / PEP 779) for Zig extensions
- Package and publish a Zig-powered Python wheel to PyPI

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Software Requirements

Students will need a free, personal GitHub account to access the courseware. Students will need permission to install Python 3.12+, Zig, and Visual Studio Code on their computers, along with Python packages including ziggy-pydust, cibuildwheel, and pytest. No separate C compiler is required — the

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Zig toolchain fills that role. If students are unable to configure a local environment, a cloud-based environment can be provided.

Training Topics

When to Drop to Native Code

- Why Python is slow for CPU-bound workloads
- CPU-bound vs IO-bound: choosing the right tool
- Profiling with cProfile and py-spy to find the real hot path
- Where Zig fits among Cython, Rust, and C/C++ for Python acceleration
- The cost model: build complexity, maintenance, and team skills

Zig Extensions via the C ABI

- Zig speaks C natively: `@cImport("Python.h")` and `extern` functions
- Module initialization and method tables from Zig
- `PyObject*` in Zig: argument parsing and building return values
- Reference counting: ownership rules, `Py_INCREF`/`Py_DECREF` discipline, and leak hunting
- Mapping Zig error unions to Python exceptions
- Allocators and lifetime at the Python boundary
- Building the shared object with `build.zig`

Ziggy Pydust

- What Pydust provides: extension packaging, comptime argument wrapping between Zig and Python types, and a pytest plugin for Zig tests
- Project layout with the Pydust template
- Defining modules, functions, and classes
- Honest status check: CPython 3.11+ support, the lag behind new Zig releases, and when to prefer the raw C API
- Alternatives in brief: setuptools-zig and thin C shims

Buffer Protocol and NumPy Interop

- The buffer protocol: zero-copy views of Python memory
- Receiving NumPy arrays in Zig as slices
- Strides, dtypes, and contiguity checks
- Writing results back without copies
- A vectorized hot loop in Zig over NumPy data

Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Cross-Compiling Wheels with zig cc and cibuildwheel

- `zig cc` as a zero-dependency drop-in cross-compiler
- Targeting manylinux (pinned glibc) and musllinux from any host
- cibuildwheel configuration for a Zig-built extension
- macOS and Windows wheels from a single CI pipeline
- The `ziglang` PyPI package: installing the toolchain with pip

Benchmarking Honestly

- pytest-benchmark methodology: warmup, variance, and fair baselines
- Comparing the same kernel in Zig, Cython, Rust, and C++
- What our High-Performance Python with Cython, High-Performance Python with Rust, and High-Performance Python with C and C++ courses each conclude — and why the boundary-crossing cost often dominates
- Avoiding benchmark theater: amortizing call overhead, realistic data sizes

The GIL and Free-Threaded Python

- What the GIL means for a Zig extension
- Releasing the GIL around CPU-bound Zig code
- Free-threaded Python (PEP 703), officially supported in 3.14 (PEP 779)
- Thread-safety obligations for Zig extension authors under free threading
- Zig’s thread-safe allocator design and shared state at the boundary

Packaging to PyPI

- Wheel metadata, abi3 considerations, and version support windows
- Source distributions when the user has (or pip-installs) Zig
- Publishing with twine and trusted publishing from GitHub Actions