CS3430 S26: Scientific Computing
Cargo Cult Programming and Why Python Scripts Still Matter

Vladimir Kulyukin
SoC – CoE – USU

Spring 2026

Motivation

In CS3430, many labs and assignments use plain `.py` scripts rather than interactive notebooks. This choice is intentional. It is not about restricting tools, but about encouraging correct scientific reasoning. This short reading explains the idea of *cargo cult programming* and why script-based workflows help prevent it.

Where the Term "Cargo Cult Programming" Comes From

The phrase *cargo cult programming* is derived from *cargo cult science*, a term popularized by physicist Richard Feynman. Dr. Feynman used it to describe situations where people carefully imitate the *appearance* of scientific work while missing its essential causal logic. The rituals look right, the structure is preserved, but the understanding that makes the process meaningful is absent.

In short:

- the form is copied,
- the motions are repeated,
- but the reasoning is missing.

Cargo Cult Programming in Practice

In computing, cargo cult programming occurs when code is used without understanding why it works. Common examples include:

- copying code from the internet because it "runs";
- executing commands without knowing what state they depend on;
- trusting outputs without checking assumptions or invariants;
- tweaking parameters until results look plausible.

The danger is not that the code crashes, but that it produces results that *appear correct* while being conceptually meaningless.

Why Interactive Notebooks Can Encourage This Behavior

Interactive notebooks are powerful tools, but they can unintentionally encourage cargo cult behavior. Typical risks include:

- running cells out of order;
- re-running some computations but not others;
- modifying code incrementally without reinitializing state;
- losing track of which assumptions produced which result.

When execution order and data dependencies are unclear, it becomes difficult to reason about correctness or reproducibility.

What Python Scripts Do Differently

A plain Python script enforces a simple but important discipline:

- execution proceeds linearly from top to bottom;
- all assumptions must be defined explicitly;
- data flow is visible and inspectable;
- results are reproducible by re-running the script.

This structure forces us to think about *process*, not just output. If a script produces a result, we can point to exactly how it was obtained.

Why This Matters in Scientific Computing

Scientific computing is not about making plots look reasonable or numbers match expectations. It is about understanding:

- how numerical error arises,
- where approximations enter,
- which assumptions matter,
- and how results depend on algorithmic choices.

Scripts make these relationships explicit. They make mistakes easier to detect, reasoning easier to follow, and results easier to trust.

Bottom Line

Using scripts is not about nostalgia or rigidity. It is about avoiding cargo cult behavior and developing habits of careful reasoning. Understanding *why* something works is more important than knowing *how* to run it.

That principle underlies CS3430 S26: Scientific Computing.