Advanced Python Programming

Facilitated by Kent State University

Duration:

- 10 Hours
- Synchronous Virtual

Instructor:

- Gregory S. DeLozier, PhD,
- Associate Professor
- Department of Computer Science

COURSE OBJECTIVES

This intensive course is designed for experienced Python programmers who want to deepen their understanding of the language's internals and master advanced techniques for building efficient, scalable, and maintainable software. Participants will explore topics from memory management to asynchronous programming, metaprogramming, concurrency, and modern deployment practices.

By the end of the course, participants will be able to:

- Understand Python's memory management including reference counting and garbage collection.
- Optimize memory usage via generators, **slots**, and specialized containers.
- Create custom iterators with **iter** and **next** protocols.
- Build generators with yield and delegation via yield from.
- Implement coroutines that consume values with .send().
- Master the event loop and cooperative multitasking.
- Run coroutines concurrently with asyncio and manage tasks effectively.
- Apply structured concurrency with TaskGroup for safe task management.
- Use functions and classes as first-class objects in metaprogramming.
- Write function and class decorators with parameters.
- Implement the descriptor protocol with **get**, **set**, **delete**.
- Create custom context managers with **enter** and **exit**.
- Understand the differences between threads vs. processes and the GIL.
- Use concurrent.futures with thread/process pools effectively.
- Implement Abstract Base Classes (ABCs) for explicit contracts.
- Apply structural subtyping with typing.Protocol (PEP 544).
- Profile and optimize Python code for production performance.
- Integrate Python with C extensions and acceleration tools like Numba and Cython.
- Package and deploy Python applications using modern tools and CI/CD pipelines.

This course is designed for experienced Python developers who want to master advanced language features and build production-ready, scalable applications.

COURSE REQUIREMENTS

- An account at GitHub
  - Codespaces permitted
- A computer with a Chrome browser
- Optional laptop software:
  - Visual Studio Code
  - Node.js and related tools
  - Python 3.x
- Setup is covered in class

COURSE OUTLINE

Here's a structured breakdown of 10 one-hour lesson topics that efficiently group related concepts while maintaining a logical progression:

Lesson 1: Python Internals & Memory Management

- Reference counting and garbage collection mechanisms
- Memory profiling with sys.getsizeof and tracemalloc
- Optimizing memory via generators, **slots**, and specialized containers

Lesson 2: Iterators, Generators & Coroutines

- Custom iterators with **iter** and **next** protocols
- Generators with yield and delegation via yield from
- Coroutines that consume values with .send()

Lesson 3: Advanced Asynchronous Programming

- The event loop and cooperative multitasking
- Running coroutines concurrently with asyncio
- Task management, cancellation, and structured concurrency with TaskGroup

Lesson 4: Metaprogramming & Decorators

- Functions and classes as first-class objects
- Writing function and class decorators (with parameters)
- Using functools.wraps and inspect for metadata and reflection

Lesson 5: Descriptors & Properties

- Descriptor protocol: **get**, **set**, **delete**
- Validation and lazy evaluation with custom descriptors
- Properties as a high-level descriptor interface

Lesson 6: Context Managers & Advanced Resource Handling

- Custom context managers with **enter** and **exit**
- contextlib tools: @contextmanager, ExitStack
- Safe handling of multiple or dynamic resources

Lesson 7: Concurrency Beyond the Basics

- Threads vs. processes (GIL and parallelism)
- concurrent.futures with thread/process pools
- Data sharing with queues and producer–consumer pipelines

Lesson 8: Data Modeling & Protocols

- Abstract Base Classes (ABCs) for explicit contracts
- Structural subtyping with typing.Protocol (PEP 544)
- Generics and bounded TypeVars for flexible, type-safe code

Lesson 9: Performance Optimization & C Extensions

- Profiling with timeit, cProfile, line_profiler
- Python-level optimizations and vectorization with NumPy
- Speedups with Numba, Cython, PyPy, and custom C extensions

Lesson 10: Packaging, Deployment & Tooling

- Packaging with pyproject.toml and wheels
- Dependency management with pip, Poetry, and Pipenv
- CI/CD pipelines for testing and publishing (GitHub Actions, PyPI)

COURSE OUTCOME

Participants will develop practical, production-level Python skills emphasizing performance, scalability, advanced design patterns, and modern deployment workflows.