Modern Concepts in Python: Spring 2026

by Eric Rying

February 20, 2026

Module 5: Software Assurance Report

1. Installing and Running the Application (pip and uv)

The application can be installed and executed using either **pip** or **uv**, depending on the user's preferred Python workflow. With pip, the user creates a virtual environment (python -m venv .venv), activates it, and installs dependencies using pip install -r requirements.txt. The application can then be run with python src/run.py or by invoking individual modules such as load_data.py or query_data.py.

Using **uv**, the workflow is similar but faster and more reproducible. After installing uv, the user runs uv venv to create an environment and uv pip install -r requirements.txt to install dependencies. uv resolves packages deterministically and caches builds, which speeds up repeated installs. Both methods support running tests via pytest and executing the application modules directly.

2. Dependency Graph Summary (5–7 sentences)

The dependency graph for Module 5 shows a small, well-structured Python application with clear separation between scraping, cleaning, database loading, querying, and the Flask web interface. The src/app package depends on routes.py, queries.py, and pages.py, which in turn rely on shared utility modules such as load_data.py and query_data.py. The scraping and cleaning modules (module_2_1/clean.py and module_2_1/scrape.py) form an isolated subgraph, meaning they can be tested and executed independently.

Database-related modules depend only on psycopg and standard library utilities, keeping the attack surface small. Flask is used only in the web layer, and no circular dependencies appear in the graph. The graph demonstrates a clean, layered architecture where data flows from scraping → cleaning → loading → querying → presentation. This structure supports high testability and minimizes the risk of unintended interactions between components.

3. SQL Injection Defenses (What Changed and Why It's Safe)

All SQL queries in Module 5 were rewritten to eliminate unsafe string formatting and to comply with psycopg's safe SQL-composition model. Previously, some queries used f-strings or concatenation, which could allow malicious input to alter SQL structure. These were replaced with sql.SQL, sql.Identifier, and sql.Placeholder objects, ensuring that table and column names are safely quoted and user-provided values are always passed as parameters.

Each query now separates **statement construction** from **execution**, using cursor.execute(stmt, params) so that values never appear directly in SQL text. Additionally, every query enforces a **safe LIMIT**, clamping user-provided limits to a fixed range (e.g., 1–100) to prevent denial-of-service attacks via unbounded result sets. These changes ensure that all dynamic SQL is safely composed, parameterized, and resistant to injection attacks.

4. Least-Privilege Database Configuration

A dedicated PostgreSQL user was created for the application with only the permissions required to operate. The account is **not** a superuser and has no ability to DROP, ALTER, or modify schema. Instead, it is granted only SELECT, INSERT, and UPDATE on the specific tables used by the application. This prevents accidental or malicious schema changes and limits the impact of a compromised application.

Database credentials are not hard-coded; instead, they are loaded from environment variables (DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASSWORD). A .env.example file documents the required variables, while the real .env file is excluded from version control. This configuration ensures that the application follows the principle of least privilege and that secrets are handled safely.

5. Requirements Met for SQL Safety

The following SQL-related requirements were fully implemented:

- **LIMIT enforced:** All queries include a LIMIT clause with safe clamping.
- **Statements and execution separated:** SQL objects are composed using psycopg's sql.SQL and executed with parameter binding.
- **Safe composition:** Dynamic identifiers use sql.Identifier, and values use sql.Placeholder.
- **Parameterization:** No f-strings, concatenation, or .format() are used for SQL.

These changes collectively ensure that the application is secure, maintainable, and compliant with Module 5's software assurance standards.

6. Why Python Packaging Matters (setup.py)

Adding setup.py makes the project installable as a Python package using pip install -e ., which ensures that Python resolves imports consistently regardless of where tests or scripts are run from. Without this, sys.path hacks or PYTHONPATH exports are needed to make imports like `from src.app import create_app` work correctly.

An editable install pins the source directory into the environment so that local runs, pytest, and CI all see the same package layout, eliminating "it works on my machine" import errors. This is especially important for testing frameworks that need to import application modules from various directory contexts.

Tools like uv can also extract requirements from setup.py when syncing environments, making reproducible installs easier to automate. The setup.py file also serves as project metadata, documenting dependencies (install_requires) and optional dev tools (extras_require) in a standardized format, improving project maintainability and collaboration.