**Sarah Lorenzen 10-1 Journal: Reflecting on Key Learnings**

**1. Discuss moments where you encountered challenges in the coursework and how you overcame them.**Security Vulnerabilities in User Input Handling

One of my most significant challenges occurred when building a Wizards Game to illustrate conditionals and OOP that accepted user data. Initially, I wrote straightforward code that directly incorporated user input into database queries and file operations. After watching the LinkedIn resource videos on Security, I realized this approach left my application vulnerable to SQL injection and path traversal attacks.

How I overcame it:

I took a systematic approach to understanding security principles. First, I researched OWASP's top vulnerabilities to understand common attack vectors. Then, I refactored my code to use parameterized queries with prepared statements, implemented input validation using regex patterns, and added sanitization layers. I also learned to use Python's secrets module for generating cryptographically strong tokens instead of the standard random module for security-sensitive operations.

Balancing Security with Performance Optimization

Another challenge was when I tried to implement both security measures and performance optimizations simultaneously. Adding encryption, extensive input validation, and security checks significantly slowed down my application's response time.

How I overcame it:

I discovered that some security measures could be cached (like password hashing parameters), while others needed to be executed every time. I used memorization for expensive validation functions.

**2. Explain how you plan to apply what you’ve learned to your future coursework, career, or personal projects.**Future Machine Learning Coursework: I'll apply secure coding practices to ML pipelines, particularly when handling sensitive training data. Input validation becomes even more critical with adversarial examples, and I'll need to consider model security alongside traditional application security.

Software Engineering Roles: The habit of writing security-conscious code from the start will differentiate me in the job market. I plan to propose security improvements during design phases, and advocate for automated security testing in CI/CD pipelines.

Personal Projects: For my personal portfolio website and API projects, I'll implement comprehensive security from day one: HTTPS, rate limiting, and secure session management.

**3. Assess how you incorporated programming principles and best practices during your software development assignments.**

Separation of Concerns: Each module had a single, well-defined responsibility. Validation was separate from logic, which was separate from data access. This made testing and auditing significantly easier.

DRY Principle (Don't Repeat Yourself): I created reusable functions to avoid duplicating logic. This meant logic updates happen in one place rather than scattered throughout the codebase.

Encapsulation for Security: Encapsulation became my primary tool for protecting sensitive data and operations. I designed classes where security-critical attributes were private (using Python's name mangling with double underscores).

Comprehensive Testing: I wrote unit tests for individual functions and used tools like unittest with coverage reporting. I also wrote tests that accounted for edge cases and attempted to exploit potential security vulnerabilities.

Input Validation and Sanitization: I consistently validated all external input. For example, when accepting file uploads, I validated file types.