Day 4: Comprehensive Guide and Notes

Focus Areas

1. **Database Programming**:
    - SQLite integration for persistent data storage.
    - CRUD operations (Create, Read, Update, Delete).
2. **Logging**:
    - Use Python's logging library for error tracking and monitoring.
    - Store logs for production monitoring and troubleshooting.
3. **Capstone Project**:
    - Manufacturing Quality Control and Reporting System.
    - Incorporate all the learned concepts.

---

1. Database Programming with SQLite

SQLite is a lightweight, file-based database suitable for small to medium applications.

Key Concepts:

1. **Connect to SQLite Database**: Use sqlite3 to connect to a database file. If the file doesn't exist, SQLite creates it.
2. **CRUD Operations**:
    - **Create**: Insert records.
    - **Read**: Query records.
    - **Update**: Modify records.
    - **Delete**: Remove records.

Example:

python

Copy code

```
import sqlite3
```

```
# Connect to SQLite
conn = sqlite3.connect('production.db')
cursor = conn.cursor()

# Create table
cursor.execute('''
CREATE TABLE IF NOT EXISTS ProductionLog (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    machine_id TEXT,
    product_type TEXT,
    units_produced INTEGER,
    production_date TEXT
)
''')

# Insert a record
cursor.execute('''
INSERT INTO ProductionLog (machine_id, product_type, units_produced,
production_date)
VALUES (?, ?, ?, ?)
''', ('MC-5673', 'Speaker', 420, '2024-01-15'))

# Query records
cursor.execute('SELECT * FROM ProductionLog')
records = cursor.fetchall()
for record in records:
    print(record)
```

```
# Close connection
conn.commit()
conn.close()
```

---

2. Logging

Python's logging module provides a powerful way to track events and errors in an application.

Key Concepts:

1. **Log Levels**:
    - DEBUG: Detailed information for debugging.
    - INFO: General information.
    - WARNING: Indications of potential issues.
    - ERROR: Errors that prevent execution.
    - CRITICAL: Severe errors causing program termination.
2. **Log to File**:
    - Store logs in a file for later analysis.

Example:

```
python
Copy code
import logging

# Configure logging
logging.basicConfig(filename='application.log', level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')

# Log events
logging.info("Application started")
logging.warning("Low inventory warning")
```

logging.error("Database connection failed")

Capstone Project: Manufacturing Quality Control and Reporting System

Objective

Develop a Python-based manufacturing subsystem for **Quality Control and Reporting**, focusing on:

1. Logging product inspection results (pass/fail, defects).
2. Validating data inputs.
3. Role-based access control.
4. Report generation.
5. GUI implementation using tkinter.

Modules and Tasks

1. Class Design

Define classes for:

- **Product**: Store product details.
- **Machine**: Track production and quality logs.
- **QualityLog**: Record inspection results.
- **Validator**: Ensure input validity.

Example:

python

Copy code

```
class Product:
    def __init__(self, product_id, name, category):
        self.product_id = product_id
        self.name = name
        self.category = category
```

```
class Machine:
    def __init__(self, machine_id, equipment_type):
        self.machine_id = machine_id
        self.equipment_type = equipment_type
        self.inspection_logs = []

    def add_quality_log(self, product, result, defects=None):
        log = {"product": product, "result": result, "defects": defects}
        self.inspection_logs.append(log)
```

---

2. GUI Development

Features:

1. **Login Screen**: Role-based access (Admin/Operator).
2. **Forms**:
    - Log inspection results.
    - View logs and generate reports.

Example:

python

Copy code

```
import tkinter as tk
from tkinter import ttk

class QualityControlApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Quality Control System")
        self.geometry("800x600")
        self.create_widgets()
```

```
    def create_widgets(self):
        tab_control = ttk.Notebook(self)
        log_tab = ttk.Frame(tab_control)
        report_tab = ttk.Frame(tab_control)

        tab_control.add(log_tab, text="Inspection Logs")
        tab_control.add(report_tab, text="Reports")
        tab_control.pack(expand=1, fill="both")

        ttk.Label(log_tab, text="Log Inspection", font=("Arial", 16)).pack(pady=10)
        ttk.Button(log_tab, text="Add Log").pack(pady=5)

        ttk.Label(report_tab, text="Generate Reports", font=("Arial", 16)).pack(pady=10)
        ttk.Button(report_tab, text="Quality Control Summary").pack(pady=5)

if __name__ == "__main__":
    app = QualityControlApp()
    app.mainloop()
```

---

3. Reporting

Generate reports such as:

1. **Quality Control Summary**:
    - Total inspections.
    - Pass and fail rates.
2. **Defect Log**:
    - Defects recorded per machine or product.

Example:

python

Copy code

```
def generate_quality_summary(machine_list):
    summary = []
    for machine in machine_list:
        total_logs = len(machine.inspection_logs)
        pass_logs = sum(1 for log in machine.inspection_logs if log["result"] == "Pass")
        fail_logs = total_logs - pass_logs


        summary.append({
            "Machine ID": machine.machine_id,
            "Total Inspections": total_logs,
            "Pass Rate": f"{(pass_logs / total_logs) * 100:.2f}%" if total_logs > 0 else "N/A",
            "Fail Rate": f"{(fail_logs / total_logs) * 100:.2f}%" if total_logs > 0 else "N/A",
        })
    return summary
```

Advanced Tools Integration

1. Streamlit:

Build web-based dashboards for visualizing quality control reports interactively.

Use Case:

- Display inspection logs and defect summaries in real time.

2. FastAPI:

Develop a REST API to expose quality control data for external integration.

Use Case:

- Provide inspection logs and summaries as JSON endpoints for external systems.

3. httpx:

Efficiently interact with APIs for submitting inspection data or retrieving product details.

Use Case:

- Fetch real-time product information from a central system.

4. OpenCV:

Analyze images from quality control processes.

Use Case:

- Detect product defects such as scratches or misalignments.

Capstone Project Workflow

Day 1: System Design

- Define class hierarchies.
- Implement Validator and exception classes.

Day 2: Core Features and GUI

- Build forms for inspection logs.
- Add role-based access control.

Day 3: Reporting and Testing

- Generate defect and quality control summaries.
- Debug and test with sample data.

Evaluation Criteria

1. **Core Functionality (40%)**:
    - Correctly log and validate inspection data.
    - Accurate report generation.
2. **GUI Implementation (30%)**:
    - User-friendly and functional interface.
3. **Data Validation (20%)**:
    - Proper handling of invalid inputs.
4. **Presentation (10%)**:
    - Clear and concise explanation of the project.

This guide provides a comprehensive roadmap for Day 4 and the Capstone Project, incorporating foundational and advanced tools to build practical applications.