# Programming Principles using Python

## Coursework: Bridge Maintenance Management System

**Weight:** 60% of module grade
**Estimated Time:** 20 hours
**Submission:** Python source code files + written report (max 1500 words)

---

## 1. Background and Context

Bridge maintenance is a critical aspect of infrastructure management. In Scotland, organisations like Transport Scotland manage thousands of bridges, requiring systematic tracking of inspections, maintenance activities, and structural conditions. This coursework simulates a simplified bridge maintenance management system where you'll implement core functionality for managing bridge assets and their inspection records.

---

## 2. Learning Outcomes Assessed

This coursework assesses your ability to:

- Design and implement classes using object-oriented programming principles
- Use appropriate data structures (lists, dictionaries, sets)
- Implement file I/O for data persistence
- Handle exceptions and validate user input
- Write clear, well-documented code following Python conventions
- Test code systematically and handle edge cases
- Apply algorithms for searching, sorting, and filtering data

---

## 3. System Requirements

You must implement a command-line application that manages bridge inspection data with the following core functionality:

### 3.1 Bridge Management

- Add new bridges to the system with unique identifiers
- Store bridge information: ID, name, location, type (e.g., suspension, arch, beam), year built
- View all bridges or search for specific bridges
- Update bridge information

## 3.2 Inspection Management

- Record new inspections for existing bridges
- Each inspection must include: date, inspector name, condition score (0-100), defects found, and recommendations
- View inspection history for any bridge
- Calculate and display statistics (average condition score, number of inspections)

## 3.3 Maintenance Prioritisation

- Identify bridges requiring urgent attention based on condition scores
- Generate a prioritised maintenance list
- Filter bridges by various criteria (condition, age, type, location)

## 3.4 Data Persistence

- Save all bridge and inspection data to files (CSV or JSON format)
- Load existing data when the program starts
- Handle file errors gracefully

## 3.5 Reporting

- Generate summary reports showing:
    - Total number of bridges in the system
    - Bridges grouped by condition category (Excellent: 80-100, Good: 60-79, Fair: 40-59, Poor: 0-39)
    - Bridges not inspected within the last 2 years
    - Average condition score across all bridges

---

# 4. Implementation Requirements

## 4.1 Object-Oriented Design

You must implement at least the following classes:

- **Bridge**: Represents a bridge with its properties and methods
- **Inspection**: Represents an inspection record
- **BridgeManagementSystem**: Main system class that manages bridges and inspections
- **FileHandler**: Handles loading and saving data

## 4.2 Code Quality Requirements

- Use meaningful variable and function names
- Include docstrings for all classes and functions
- Add comments for complex logic
- Follow PEP 8 style guidelines
- Implement proper error handling with try-except blocks
- Validate all user inputs

## 4.3 Required Python Features

Your implementation must demonstrate use of:

- Classes and objects with appropriate methods
- Class attributes and instance attributes
- Lists and dictionaries for data management
- File I/O (reading and writing)
- Exception handling
- At least one list comprehension
- At least one function that uses *args or **kwargs (where appropriate)
- String formatting (f-strings recommended)

## 4.4 Menu System

Implement a text-based menu system that allows users to:

1. Add a new bridge
2. Record a new inspection
3. View all bridges
4. View bridge inspection history
5. Generate maintenance priority list
6. Generate summary report
7. Search/filter bridges
8. Save and exit

---

# 5. Deliverables

## 5.1 Code Submission

Submit the following Python files:

- `bridge.py` - Bridge class implementation
- `inspection.py` - Inspection class implementation
- `bridge_management_system.py` - Main system class
- `file_handler.py` - File I/O operations
- `main.py` - Menu interface and program entry point
- `test_data.py` (optional) - Script to generate sample test data

## 5.2 Written Report (Max 1500 words)

Your report should include:

1. **Design Overview** (400 words)
    - Class diagram or description of your class structure
    - Explanation of key design decisions
    - Data structures chosen and why
2. **Implementation Challenges** (400 words)
    - Difficult aspects of the implementation
    - How you overcame challenges
    - Alternative approaches considered
3. **Testing Strategy** (400 words)
    - How you tested your code
    - Edge cases considered
    - Example test cases with expected vs actual results
4. **Reflection** (300 words)
    - What you learned from this coursework
    - What you would improve given more time
    - How this relates to real-world systems

---

# 6. Assessment Criteria

| Criterion | Marks | Description |
|---|---|---|
| **Object-Oriented Design** | 20 | Appropriate use of classes, encapsulation, and methods |
| **Core Functionality** | 25 | All required features working correctly |
| **Data Persistence** | 10 | Correct file I/O with error handling |
| **Code Quality** | 15 | Readability, documentation, PEP 8 compliance |
| **Error Handling** | 10 | Input validation and exception handling |
| **Written Report** | 20 | Clear explanation of design, testing, and reflection |
| **Total** | **100** | |

---

# 7. Marking Scheme Details

## Object-Oriented Design (20 marks)

- Excellent (17-20): Well-designed classes with clear responsibilities, good encapsulation, appropriate use of inheritance if applicable
- Good (13-16): Functional class design with minor issues in structure or encapsulation
- Satisfactory (9-12): Basic class structure but poor separation of concerns or weak encapsulation
- Poor (0-8): Minimal or inappropriate use of OOP principles

## Core Functionality (25 marks)

- Excellent (21-25): All features implemented and working correctly with no bugs
- Good (16-20): Most features working with minor bugs or missing edge case handling
- Satisfactory (11-15): Core features working but significant functionality missing
- Poor (0-10): Many features missing or not working

## Data Persistence (10 marks)

- Excellent (9-10): Robust file handling with proper error handling and data validation
- Good (7-8): File I/O works with minor issues in error handling
- Satisfactory (5-6): Basic file operations but poor error handling
- Poor (0-4): File operations not working or missing

## Code Quality (15 marks)

- Excellent (13-15): Exceptionally clear code with comprehensive documentation
- Good (10-12): Well-written code with good documentation
- Satisfactory (7-9): Readable code but inconsistent documentation
- Poor (0-6): Poor naming, lacking documentation, hard to understand

## Error Handling (10 marks)

- Excellent (9-10): Comprehensive input validation and graceful error handling
- Good (7-8): Most errors caught and handled appropriately
- Satisfactory (5-6): Basic error handling but some cases missed
- Poor (0-4): Minimal or no error handling

## Written Report (20 marks)

- Excellent (17-20): Comprehensive, well-structured report with clear explanations
- Good (13-16): Good coverage of required sections with minor gaps
- Satisfactory (9-12): Adequate coverage but lacking depth or clarity
- Poor (0-8): Incomplete or poorly written report

---

# 8. Sample Data

To help you test your system, here are example bridges you might include:

| Bridge ID | Name | Location | Type | Year Built |
|---|---|---|---|---|
| B001 | Forth Bridge | Queensferry | Cantilever | 1890 |
| B002 | Forth Road Bridge | Queensferry | Suspension | 1964 |
| B003 | Queensferry Crossing | Queensferry | Cable-stayed | 2017 |
| B004 | Erskine Bridge | Erskine | Cable-stayed | 1971 |
| B005 | Kingston Bridge | Glasgow | Arch | 1970 |

Sample inspection data:

- Bridge B001, Date: 2024-03-15, Inspector: John Smith, Score: 72, Defects: "Minor paint deterioration on south tower", Recommendations: "Schedule repainting in next 2 years"
- Bridge B003, Date: 2024-09-20, Inspector: Sarah Jones, Score: 95, Defects: "None observed", Recommendations: "Continue routine monitoring"

---

# 9. Extensions (Optional - for students seeking distinction)

If you complete the core requirements and want additional challenge, consider implementing:

1. **Advanced Filtering**: Complex queries combining multiple criteria (e.g., "Show all suspension bridges built before 1980 with condition score below 60")
2. **Data Visualisation**: Generate simple text-based charts showing condition score distributions or inspection frequency
3. **Inspection Scheduling**: Automatically suggest which bridges need inspection based on time since last inspection
4. **Defect Categories**: Classify defects into categories (structural, cosmetic, safety-critical) and analyse by category
5. **User Authentication**: Simple username/password system with different access levels
6. **Bulk Data Import**: Import bridge/inspection data from existing CSV files

**Note**: Extensions are not required and will only be considered if core functionality is excellent.

---

# 10. Academic Integrity

- This is an individual assignment. Do not share code with other students.
- You may use Python documentation and standard learning resources.
- If you use any external code snippets, cite them in comments and your report.

- Using AI tools (ChatGPT, Copilot, etc.) to generate substantial portions of code is not permitted.
- Plagiarism will result in penalties according to university policy.

---

# 11. Submission Instructions

1. Create a ZIP file containing:
    - All Python source code files
    - A `README.txt` file with instructions to run your program
    - Your written report as a PDF
    - Any data files needed to run your program
2. Name your ZIP file: `StudentID_BridgeSystem.zip`
3. Submit via the module's learning management system by the deadline
4. Late submissions will be penalised according to university policy

---

# 12. Getting Help

- Attend lab sessions to get help with specific implementation issues
- Post general questions on the module forum (don't share code)
- Review lecture materials on OOP, file I/O, and exception handling
- Consult Python documentation: https://docs.python.org/3/

---

# 13. Frequently Asked Questions

**Q: What date format should I use for inspections?**
A: Use ISO format (YYYY-MM-DD). Python's `datetime` module is helpful.

**Q: Should I use CSV or JSON for data storage?**
A: Either is acceptable. CSV is simpler; JSON handles nested data better.

**Q: How should I handle the "not inspected in 2 years" requirement?**
A: Compare inspection dates with the current date. The `datetime` module can help with date arithmetic.

**Q: Can I use external libraries?**
A: Only standard library modules (datetime, json, csv, etc.). No third-party packages.

**Q: What if a bridge has no inspections?**
A: Handle this case gracefully - it shouldn't cause errors.

**Q: How detailed should my menu system be?**
A: It should be functional and user-friendly, but doesn't need to be fancy. Clear prompts and error messages are important.

---

**All the best with your coursework!**