Contents

Chapter I

Foreword

Welcome to the Cyber Archives, future Data Archivist!

In the year 2087, humanity's greatest treasure isn't gold or diamonds—it's **data**. Every piece of digital knowledge, from ancient memes to quantum algorithms, is stored in the vast Cyber Archives. But data is fragile. Without proper preservation, it vanishes into the digital void forever.

That's where you come in. As a Data Archivist, you're humanity's guardian against information entropy. Your mission: master the ancient arts of **file operations** to ensure no knowledge is ever lost.

The Archives contain millions of data fragments scattered across different storage systems. Some are encrypted in ancient formats, others are corrupted by time. Your tools are simple but powerful: the ability to **open** data vaults, **read** their contents, **write** new archives, and, most crucially, handle the unexpected without losing precious information.

Every great archivist started where you are now—learning to communicate with the storage systems, understanding the three sacred streams of data flow, and mastering the protective rituals that keep data safe. The Archives are counting on you!

Chapter II

AI Instructions

Context

During your learning journey, AI can assist with many different tasks. Take the time to explore the various capabilities of AI tools and how they can support your work. However, always approach them with caution and critically assess the results. Whether it's code, documentation, ideas, or technical explanations, you can never be completely sure that your question was well-formed or that the generated content is accurate. Your peers are a valuable resource to help you avoid mistakes and blind spots.

Main message

- Use AI to reduce repetitive or tedious tasks.
- Develop prompting skills — both coding and non-coding — that will benefit your future career.
- Learn how AI systems work to better anticipate and avoid common risks, biases, and ethical issues.
- Continue building both technical and power skills by working with your peers.
- Only use AI-generated content that you fully understand and can take responsibility for.

Learner rules:

- You should take the time to explore AI tools and understand how they work, so you can use them ethically and reduce potential biases.
- You should reflect on your problem before prompting — this helps you write clearer, more detailed, and more relevant prompts using accurate vocabulary.
- You should develop the habit of systematically checking, reviewing, questioning, and testing anything generated by AI.
- You should always seek peer review — don't rely solely on your own validation.

Phase outcomes:

- Develop both general-purpose and domain-specific prompting skills.
- Boost your productivity with effective use of AI tools.
- Continue strengthening computational thinking, problem-solving, adaptability, and collaboration.

Comments and examples:

- You'll regularly encounter situations — exams, evaluations, and more — where you must demonstrate real understanding. Be prepared, keep building both your technical and interpersonal skills.
- Explaining your reasoning and debating with peers often reveals gaps in your understanding. Make peer learning a priority.
- AI tools often lack your specific context and tend to provide generic responses. Your peers, who share your environment, can offer more relevant and accurate insights.
- Where AI tends to generate the most likely answer, your peers can provide alternative perspectives and valuable nuance. Rely on them as a quality checkpoint.

✓ Good practice:

> I ask AI: "How do I test a sorting function?" It gives me a few ideas. I try them out and review the results with a peer. We refine the approach together.

✗ Bad practice:

> I ask AI to write a whole function, copy-paste it into my project. During peer-evaluation, I can't explain what it does or why. I lose credibility — and I fail my project.

✓ Good practice:

> I use AI to help design a parser. Then I walk through the logic with a peer. We catch two bugs and rewrite it together — better, cleaner, and fully understood.

✗ Bad practice:

> I let Copilot generate my code for a key part of my project. It compiles, but I can't explain how it handles pipes. During the evaluation, I fail to justify and I fail my project.

Chapter III

Introduction

Welcome to the Cyber Archives Training Program!

The Head Archivist has assigned you to the Digital Preservation Division. Your training consists of five critical missions, each building your expertise in data handling:

- **Mission 0**: Ancient Text Recovery - Retrieve data from old storage units
- **Mission 1**: Archive Creation - Establish new data preservation protocols
- **Mission 2**: Stream Management - Master the three data channels
- **Mission 3**: Vault Security - Implement failsafe storage procedures
- **Mission 4**: Crisis Response - Handle corrupted archives and access failures

Each mission simulates real archival scenarios. The systems you'll work with may seem simple, but they form the foundation of all digital preservation. Master these fundamentals, and you'll be ready to protect humanity's digital heritage.

ARCHIVE PROTOCOL: All operations must follow strict data preservation standards. Corrupted archives mean lost knowledge forever.

Chapter IV

Training Resources

The Archives provide essential training materials in the `attachments/` directory:

- **Tools Archive**: Extract `data-generator-tools.tar.gz` to access the training toolkit
- **Data Generator**: Execute `python3 data_generator.py` to create training files required for **Exercise 0** (ancient_fragment.txt) and **Exercise 4** (standard_archive.txt and other test files)
- **Sample Data**: Reference examples available in `data_samples/` to understand expected data formats and structures
- **JSON Configuration**: The file `sample_data.json` contains metadata and test scenario information for reference purposes only

Extract the tools archive first: `tar -xzf data-generator-tools.tar.gz`
Then generate your training data before beginning the exercises. The Archives' security protocols require specific file structures and content formats for proper evaluation.

Chapter V

Archivist Training Protocol

Welcome to the Cyber Archives Training Program. As a Data Archivist trainee, you'll master the ancient arts of file manipulation and resource management that have protected digital knowledge for centuries.

V.1 Core Principles

The Archivist's Code: Every file operation must be secure, every resource properly managed, and every error gracefully handled. The Archives have survived digital catastrophes because archivists follow these sacred protocols.

The examples provided in each exercise demonstrate the expected behavior. Study them carefully-they contain the wisdom of master archivists who came before you.

V.2 Technical Requirements

Your training programs must demonstrate mastery of:

- **Python 3.10+**: All programs must be written in Python 3.10 or later
- **Code Quality**: Your code must respect the flake8 linter standards
- **Type Hints**: All functions and methods must include type hints
- **File Operations**: Reading from and writing to storage vaults
- **Stream Management**: Proper use of input, output, and alert channels
- **Context Managers**: The sacred `with` protocol for resource safety
- **Error Handling**: Crisis response using `try-except` protocols

V.3 Authorized Libraries

For data processing operations, you are authorized to use:

- **Standard library**: `sys` module for stream management
- **Built-in functions**: `open()`, `read()`, `write()`, `close()`, `print()`, `input()`

Failure to properly `close` file handles has caused data corruption throughout history. Always use the with statement when accessing storage vaults-it's not just good practice, it's survival.

V.4 Output Specifications

Each exercise provides terminal examples showing the expected output format. While the core structure must be maintained, you may customize messages to reflect your understanding of the Archives' operations, as long as the essential information is preserved.

Pro Archivist Tip: The best archivists add their own personality to the system messages while maintaining professional standards. Make the Archives feel alive with your unique touch!

What makes a good Data Archivist? One who understands that every file is a piece of history worth preserving, and every operation is a responsibility to future generations.

Chapter VI

Exercise 0: Ancient Text Recovery

	Exercise0
ft_ancient_text	
Directory: *ex*0/	
Files to Submit: `ft_ancient_text.py`	
Authorized: `open(), read(), close(), print()`	

Mission Briefing: The Archives have discovered an ancient data fragment in Storage Vault 7. Your first assignment as a Data Archivist is to recover this precious information before it degrades further. Legend says it contains wisdom from the era when developers actually wrote documentation!

Data Source: Use the provided data generator to create your test file:

Run:
`python3 data_generator.py to create ancient_fragment.txt` with the required data.
Your program must read from this exact filename; the Archives security protocols are very specific about file naming conventions.

Archive Recovery Protocol: Access the storage unit containing `ancient_fragment.txt`, extract all preserved data, and display the recovery process. Think of it as digital archaeology—you're uncovering lost wisdom from the depths of the storage matrix.

If the vault is inaccessible, your program should display: ERROR: Storage vault not found. Run data generator first. Remember: a good archivist always checks if the vault exists before attempting access. Trying to read non-existent files is like trying to open a door that isn't there-it never ends well.

Expected Output: Your program must display the system header, vault access status, recovered data with proper formatting, and completion confirmation. The terminal example shows the exact format, but you may add your own archivist personality to the messages as long as the core information is preserved.

Recovery log:

```
$> python3 ft_ancient_text.py
=== CYBER ARCHIVES - DATA RECOVERY SYSTEM ===

Accessing Storage Vault: ancient_fragment.txt
Connection established...

RECOVERED DATA:
[FRAGMENT 001] Digital preservation protocols established 2087
[FRAGMENT 002] Knowledge must survive the entropy wars
[FRAGMENT 003] Every byte saved is a victory against oblivion

Data recovery complete. Storage unit disconnected.
```

What happens to the storage system if connections aren't properly closed? Why is the disconnect protocol critical?

Chapter VII

Exercise 1: Archive Creation

s

	Exercise1
ft_archive_creation	
Directory: *ex*1/	
Files to Submit: `ft_archive_creation.py`	
Authorized: `open(), write(), close(), print()`	

Mission Briefing: Excellent work on the data recovery! The Head Archivist is impressed. Your next assignment: establish a new preservation protocol by creating fresh archive entries. Time to make history instead of just reading it!

Archive Creation Protocol: Establish a new storage unit named `new_discovery.txt` and inscribe three critical data entries: quantum algorithm breakthrough information, performance improvement metrics (347% efficiency gain), and archivist identification. Think of it as writing a digital time capsule for future generations.

Remember that preservation mode creates new archives or replaces existing ones. Unlike reading, writing is permanent-once you seal the vault, the data becomes part of the eternal Archives.

Expected Output: Your program must display the system header, storage unit initialization, data inscription with numbered entries, and completion confirmation. The terminal example shows the expected format, but feel free to add your own archival flourishes.

Be careful with write operations! In the real Archives, accidentally overwriting historical data is considered a crime against knowledge itself. Always double-check your file names.

Archive log:

```
$> python3 ft_archive_creation.py
=== CYBER ARCHIVES - PRESERVATION SYSTEM ===

Initializing new storage unit: new_discovery.txt
Storage unit created successfully...

Inscribing preservation data...
[ENTRY 001] New quantum algorithm discovered
[ENTRY 002] Efficiency increased by 347%
[ENTRY 003] Archived by Data Archivist trainee

Data inscription complete. Storage unit sealed.
Archive 'new_discovery.txt' ready for long-term preservation.
```

What's the critical difference between extraction mode ('r') and preservation mode ('w')? Why is this distinction vital for archivists?

Chapter VIII

Exercise 2: Stream Management

	Exercise2
ft_stream_management	
Directory: *ex2/*	
Files to Submit: `ft_stream_management.py`	
Authorized: `sys, sys.stdin, sys.stdout, sys.stderr, input(), print()`	

Mission Briefing: Outstanding archive work! The Head Archivist has a new challenge for you. The Archives operate through three sacred data channels that have been active since the founding of digital civilization. These channels are older than the internet itself-they're the digital equivalent of ancient trade routes.

Communication Protocol: Access the three sacred data channels of the Archives—input stream, standard channel, and alert channel. Collect archivist identification and status information, then demonstrate proper channel separation by routing different message types to their appropriate streams.

Think of the streams like different communication frequencies: stdin for receiving messages, stdout for normal broadcasts, and stderr for emergency alerts. Each has its purpose in the grand communication network.

Expected Output: Your program must display the system header, collect user input for archivist ID and status report, then output messages using the correct streams: standard messages via `sys.stdout` and alerts via `sys.stderr`. Complete with a successful communication test confirmation.

Never mix your streams! Sending alerts through the standard channel is like shouting "FIRE!" in a library whisper-it defeats the purpose and confuses everyone.

Communication log:

```
$> python3 ft_stream_management.py
=== CYBER ARCHIVES - COMMUNICATION SYSTEM ===

Input Stream active. Enter archivist ID: ARCH_7742
Input Stream active. Enter status report: All systems nominal

[STANDARD] Archive status from ARCH_7742: All systems nominal
[ALERT] System diagnostic: Communication channels verified
[STANDARD] Data transmission complete

Three-channel communication test successful.
```

Why do the Archives maintain separate channels for standard data and alerts? What could happen if these streams were mixed?

Chapter IX

Exercise 3: Vault Security

	Exercise3
ft_vault_security	
Directory: *ex3/*	
Files to Submit: `ft_vault_security.py`	
Authorized: `open()`, `read()`, `write()`, `print()`	

This exercise requires the use of the 'with' statement (context manager) to ensure proper file handling. The 'with' statement automatically closes files even if errors occur, preventing data corruption and resource leaks.

Mission Briefing: Impressive communication skills! The Head Archivist has noticed your potential and is promoting you to Vault Security operations. This is where the real archivists prove themselves. Welcome to the big leagues-where one mistake can corrupt centuries of knowledge.

The Archives have learned from centuries of data loss. The ancient **with** protocol was developed after the Great Data Corruption of 2089, when thousands of archives were lost because connections weren't properly closed during system failures.

The Sac red Protocol: When you open a vault with the **with** statement, three things happen automatically: the vault opens, your operations execute safely within its protective barrier, and the vault seals itself-even if something goes wrong. It's like having a digital bodyguard for your files.

Vault Security Protocol: Implement secure file operations using the **with** statement for both reading classified data and preserving new information. Your program should demonstrate automatic vault sealing regardless of operation success or failure, with professional security logging throughout.

This is the essence of secure resource management: acquire, use, release-guaranteed. Without the with statement, you're basically leaving vault doors open in a digital hurricane. Don't be that archivist.

Security log:

```
$> python3 ft_vaul_security.py
=== CYBER ARCHIVES - VAULT SECURITY SYSTEM ===

Initiating secure vault access...
Vault connection established with failsafe protocols

SECURE EXTRACTION:
[CLASSIFIED] Quantum encryption keys recovered
[CLASSIFIED] Archive integrity: 100%

SECURE PRESERVATION:
[CLASSIFIED] New security protocols archived
Vault automatically sealed upon completion

All vault operations completed with maximum security.
```

How does the **with** protocol prevent data corruption? What is the RAII principle and why is it crucial for vault security?

Chapter X

Exercise 4: Crisis Response

	Exercise4
ft_crisis_response	
Directory: *ex4/*	
Files to Submit: `ft_crisis_response.py`	
Authorized: `open()`, `read()`, `write()`, `print()`	

This exercise requires the use of both the 'with' statement for safe file handling and try/except blocks for error handling. You must handle FileNotFoundError, PermissionError, and other exceptions gracefully while ensuring files are properly closed even when errors occur.

Mission Briefing: Exceptional vault security work! clearance level. The Head Archivist has one final test: Crisis Response operations. This is the ultimate trial-where archivists prove they can handle the chaos of real-world data disasters.

Data Preparation:

Run: `python3 tools/data_generator.py` to create test files including `standard_archive.txt` and other test files. Your program will test access to various files to simulate different crisis scenarios.

Crisis Response Protocol: Develop a comprehensive crisis management system that implements a crisis handler function for archive operations. The system must manage different types of archive access failures gracefully using failsafe protocols combined with the **with** statement to prevent data corruption during errors.

In the real Archives, crises happen daily: corrupted files, missing data, security breaches. A master archivist doesn't panic-they respond with precision and grace, ensuring the system remains stable no matter what chaos unfolds.

Crisis Categories: Handle missing archives in the storage matrix, security protocol violations, unexpected system anomalies, and successful operations. Test with access attempts to non-existent archives, security-restricted vaults, and standard archive recovery operations.

Expected Output: Your program must display crisis alerts for each access attempt, appropriate response messages based on error types (`FileNotFoundError`, `PermissionError`, or other exceptions), status confirmations, and overall security completion. The terminal example shows the expected format—follow it precisely for consistency across the Archives network.

Crisis log:

```
$> python3 ft_crisis_response.py
=== CYBER ARCHIVES - CRISIS RESPONSE SYSTEM ===

CRISIS ALERT: Attempting access to 'lost_archive.txt'...
RESPONSE: Archive not found in storage matrix
STATUS: Crisis handled, system stable

CRISIS ALERT: Attempting access to 'classified_vault.txt'...
RESPONSE: Security protocols deny access
STATUS: Crisis handled, security maintained

ROUTINE ACCESS: Attempting access to 'standard_archive.txt'...
SUCCESS: Archive recovered - ``Knowledge preserved for humanity''
STATUS: Normal operations resumed

All crisis scenarios handled successfully. Archives secure.
```

What are the most dangerous threats to digital archives? How does proper crisis response prevent data loss and maintain system stability?

Chapter XI

Turn in and Submission

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your files to ensure they are correct.

During evaluation, you may be asked to explain file operations, demonstrate error handling, or show how the with statement works. Make sure you understand the concepts behind each exercise.

You need to return only the files requested by the subject of this project. Focus on clean, simple code that clearly demonstrates your understanding of file operations.