Table of contents

AI-Assisted Python Programming - Final Exam

Duration: 3 hours

Allowed Resources: Course materials, internet, any AI assistant

Academic Integrity: You must document all AI interactions. Collaboration with other students is prohibited.

COURSE CONSTRAINTS - CRITICAL

All code must use ONLY techniques taught during this semester.

Any use of concepts not covered in our course will result in zero marks for that section.

GOOGLE COLAB + GITHUB WORKFLOW

Required Setup:

1. Create a **private GitHub repository** named `final-exam-[yourname]`
2. Work in **Google Colab** using our standard course setup
3. **Save to GitHub** after each major section completion
4. Include **doctest examples** for all functions (our standard testing approach)
5. Final submission: **Share private repo** with instructor account

Time Management Guide

- **Section 1:** 45 minutes (Iterative Prompting)
- **Section 2:** 40 minutes (Debug & Correct)
- **Section 3:** 50 minutes (Debug & Refine)
- **Section 4:** 45 minutes (Implement & Reflect)

Submission Requirements

GitHub Repository Setup:

1. Create **private repository**: `final-exam-[yourname]`

2. Work in **Google Colab** with our standard course environment

3. **Repository structure:**

```
final-exam-[yourname]/
   Final_Exam.ipynb (main notebook)
   conversation_log.txt
   task_manager.py (if created separately)
   weather_fetcher.py (if created separately)
   README.md (brief summary)
```

4. **Save to GitHub** after completing each section

5. **Final submission:** Share private repo with instructor GitHub account

6. **Tag final version** as `v1.0` using GitHub interface

Notebook Cell Structure (Follow Course Standard)

```
# Section X.Y - [Description]
# Course constraint: [Week X concept being used]

def function_name(parameters):
    """
    Brief description

    >>> function_name(test_input)
    expected_output
    """
    # Your implementation

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Conversation Log Format

```
SECTION X - PROMPT Y:
[Your prompt here]

AI RESPONSE:
[Relevant AI response - you may abbreviate long responses]

COURSE CONSTRAINT CHECK:
[Note any times you had to modify AI suggestions to fit course constraints]

---
```

Assessment Rubric

Section	Component	Marks	Criteria
1	Initial Prompt & Response	8	Clear problem statement (3), Complete request (3), AI response quality (2)
	Two Refinements	12	Meaningful improvements (6), Specific clarifications (6)
	Critical Analysis + Course Refs	10	Analysis depth (5), Course material references (5)
2	Error Identification	10	Completeness (6), AI interaction quality (4)
	Fix & Manual Rewrite	15	Correct fixes (8), Course-appropriate complexity (4), Hand-coding accuracy (3)
3	Issue Analysis + Course Connection	6	Identifies 3+ issues (3), Course concept application (3)
	Function Refinement	10	Working code (6), Week 8 error handling only (4)
	Comparison Analysis	4	Similarities/differences (2), Pros/cons (2)
4	Manual Implementation	15	Functional code (8), Course constraints followed (4), Clear commenting (3)
	Course-Connected Reflection	10	Workflow analysis (3), Course learning connections (4), Future application (3)
Total		**100**	

Section 1: Iterative Prompt Engineering (30 marks)

Time: 45 minutes

You're designing a CLI **Task Manager** that allows users to add, list, and remove tasks stored in memory.

Course Constraint Reminder

Your pseudocode must reflect **only techniques taught through to Week 10:**

- Basic lists and dictionaries for data storage
- Simple functions with parameters
- Basic file operations (if needed for persistence)
- While loops for menu systems
- Try/except for basic error handling only

1.1 Initial Prompt & Pseudocode (8 marks)

Task: Draft your first AI prompt requesting a structured pseudocode outline using the planning methodology from our course.

Required:

- Your solution must only use techniques taught in this semester
- **Reference:** State which specific week/topic taught the planning approach you're using

Deliverable:

- Quote your exact prompt and the AI's complete pseudocode response
- **Course Reference:** "I used the planning method from [specific week/topic]"

1.2 Two Prompt Refinements (12 marks)

Task: Refine your original prompt **twice**, each time adding specificity or addressing overlooked requirements.

Deliverable: For each refinement:

- Quote your refined prompt
- Quote AI's updated pseudocode
- **Course Reference:** "This refinement applied [concept] from [specific week/topic]"

1.3 Critical Analysis with Course References (10 marks)

Task: Write 150 words analyzing your prompt evolution, **explicitly referencing course materials**:

Required references:

- Reference specific weeks/topics that influenced your approach
- Explain how your refinements applied course concepts
- Connect your final pseudocode to specific course learning

Format: “My refinements applied [specific concept] from [week/topic] because…”

Section 2: Debug & Correct with AI (25 marks)

Time: 40 minutes

The Buggy Code (Contains errors typical of Week 6 skill level)

```
# broken_task_manager.py
tasks = []

def add_task(task):
    tasks.append(task)
    print(f"Added: {task}")

def remove_task(index):
    if index < len(tas):  # Error 1: typo
        removed = tasks[index]
        del tasks[index]
        print(f"Removed: {removed}")
    else:
        print("Invalid index!")

def list_tasks():
    if not tasks:
        print("No tasks available.")
    else:
        for i, t in enumerate(task):  # Error 2: wrong variable
```

```
            print(f"{i+1}: {t}")

def main():
    add_task("Buy milk")
    add_task("Pay bills")
    add_task("Walk dog")
    list_tasks()
    remove_task(1)
    list_task()  # Error 3: wrong function name

if __name__ == "__main__":
    main()
```

2.1 Error Identification (10 marks)

Task:

1. Prompt your AI to identify errors in this code
2. Quote your prompt and the AI's complete error analysis
3. **Course Reference:** "I applied the debugging approach from [specific lecture/worksheet]"

2.2 Fix & Manual Rewrite (15 marks)

Task:

1. Prompt your AI to provide corrected code
2. Quote your prompt and AI's response
3. **Manually rewrite** the entire corrected script in a **new Colab cell**
4. Add **doctest examples** for each function
5. Include comments explaining your fixes

Required:

- Use only techniques taught in our course
- **Course Reference:** "My error handling approach comes from [specific lecture/worksheet]"
- **Course Reference:** "My testing approach follows [specific lecture/worksheet]"

Section 3: Debug & Refine WeatherWise API (20 marks)

Time: 50 minutes

The Current Implementation (Reflects Week 8 learning level)

```python
import requests

def safe_weather_data_fetch(city):
    """Fetch weather data for a city from wttr.in API - Week 8 version"""
    try:
        url = f"http://wttr.in/{city}?format=j1"
        response = requests.get(url)
        data = response.json()

        weather_info = {
            'city': city,
            'temperature': data['current_condition'][0]['temp_C'],
            'wind_speed': data['current_condition'][0]['windspeedKmph'],
            'description': data['current_condition'][0]['weatherDesc'][0]['value']
        }
        return weather_info
    except:
        return "Error occurred"

def ideal_safe_weather_data_fetch(city):
    """Improved version using Week 8 error handling concepts"""
    try:
        # Basic input validation - Week 6 concept
        if not city:
            print("Error: City name cannot be empty")
            return None

        url = f"http://wttr.in/{city}?format=j1"
        response = requests.get(url)
        data = response.json()

        # Safe data extraction with basic error checking
        try:
            current = data['current_condition'][0]
            weather_info = {
                'city': city,
                'temperature': current['temp_C'],
                'wind_speed': current['windspeedKmph'],
                'description': current['weatherDesc'][0]['value']
            }
            return weather_info
        except:
            print("Error: Could not extract weather data from response")
            return None

    except:
        print("Error: Could not connect to weather service")
        return None
```

Required references:

- Reference specific weeks/topics that influenced your approach
- Explain how your refinements applied course concepts
- Connect your final pseudocode to specific course learning

3.1 Issue Analysis with Course Connection (6 marks)

Task:

1. List 3 problems with the current `safe_weather_data_fetch()` function
2. **For each problem**, reference which **course week/concept** provides the solution approach
3. Prompt AI: "Review this function using only error handling techniques taught in an introductory programming course (basic try/except only)."

3.2 Function Refinement with Course Constraints (10 marks)

Task:

1. Create `refined_safe_weather_data_fetch(city)` in a **new Colab cell** using **only Week 8 error handling techniques**:
    - Basic try/except blocks (no specific exception types)
    - Simple input validation with if statements
    - Print statements for error messages (no logging module)
    - Return None or dictionary (consistent with Week 8 examples)
    - **Include doctest examples** following Week 9 format

2. **Course Constraint Enforcement:** Your code must look like something from Week 8 lab exercises - simple, clear, using only basic concepts.

Required doctest format:

```
def refined_safe_weather_data_fetch(city):
    """
    Fetch weather data with basic error handling - Week 8 style

    >>> refined_safe_weather_data_fetch("")
    Error: City name cannot be empty
    >>> refined_safe_weather_data_fetch("InvalidCity123")  # doctest: +SKIP
    Error: Could not connect to weather service
    """
    # Your implementation
```

Forbidden Advanced Techniques:

- Specific exception types (ValueError, KeyError, etc.)
- Multiple except blocks
- Exception chaining or custom exceptions
- Advanced string formatting beyond f-strings

Remember: Save to GitHub after completing this function.

3.3 Comparison Analysis (4 marks)

Compare your refined version with the provided ideal version:

- **2 Similarities:** Basic approaches both versions share
- **2 Differences:** How they handle complexity differently

- **1 Course Connection:** Which **Week 8 concept** your version demonstrates best
- **1 Improvement Area:** Something you'd change after reviewing **Textbook Chapter 6**

Section 4: Manual Implementation & Course Reflection (25 marks)

Time: 45 minutes

4.1 Manual Implementation with Course Constraints (15 marks)

Task: Using your final pseudocode from Section 1, implement these functions **manually in Colab cells using only Week 4-6 techniques**:

```
def display_menu():
    """
    Display menu options - use Week 3 formatting techniques only

    >>> display_menu()  # doctest: +SKIP
    1. Add task
    2. List tasks
    3. Remove task
    4. Quit
    """
    # Your implementation here
    pass

def get_user_choice():
    """
    Get and validate user choice - Week 6 input validation style

    >>> # This function requires user input, so we'll test manually
    >>> # get_user_choice()  # doctest: +SKIP
    """
    # Your implementation here
    pass

def main():
    """Main programme loop - Week 5 loop patterns only"""
    # Your implementation here - no doctest needed for main
    pass
```

Course Constraints:

- Use only basic input() and print() for user interaction
- Simple while loops with basic conditions
- Basic if/else for validation (no complex logic)
- **Include doctest for testable functions** (Week 9 requirement)
- Variable names and style matching our course examples
- Comments referencing specific course concepts used
- **Save to GitHub** after implementing each function

Complexity Level: Should match Lab 5 exercise difficulty - functional but not sophisticated.

doctest Requirements:

- At least 2 test cases for each testable function
- Use `# doctest: +SKIP` for functions requiring user input
- Follow Week 9 doctest formatting standards

4.2 Course-Connected Reflection (10 marks)

Write exactly 200 words addressing these specific questions with **required course references**:

1. Workflow Balance with Course Learning (70 words): How did you apply concepts from the lecture slides? Reference specific strategies from dividing work between human planning and AI assistance.

2. Course Concept Application (70 words): Identify one moment where you applied **"Error Handling Mindset"** and one where you used **"Defensive Programming" approach**. How do these connect to your experience in testing?

3. Learning Transfer (60 words): How will you apply the **iterative refinement process** to your final project? Reference one specific technique from **Assignment 2 feedback** that you'll improve using this approach.

Required Format: - Bold the specific week/document references - Use our course terminology (not generic programming terms) - Connect to your actual course experience, not hypothetical scenarios

---

Course Material Integration Requirements

Required References Throughout Exam:

Students must demonstrate familiarity with: - "Planning Before Coding" methodology - Input validation and defensive programming - Basic exception handling patterns - Function design principles - Error handling fundamentals - Previous project experience - Debugging practice session

Course Terminology to Use:

- "Defensive programming" (not "robust coding")
- "Iterative design" (not "agile development")
- "Basic exception handling" (not "comprehensive error management")
- "Input validation" (not "data sanitisation")

Acceptable Skill Level Indicators:

- Code that works but isn't highly optimised
- Basic variable names (not overly sophisticated)
- Simple logic flow matching course examples
- Comments that reference our specific debugging process
- Error handling that matches lab complexity

Troubleshooting Guide

Google Colab + GitHub Setup:

1. **Create private repo** on GitHub: `final-exam-[yourname]`
2. **In Colab:** File → Save a copy in GitHub → Select your repo
3. **Regular saves:** Ctrl+S, then File → Save to GitHub
4. **Final submission:** Share repo with instructor GitHub account

Course Constraint Violations: If AI suggests advanced techniques:

```
"Please revise this to use only basic Python concepts suitable for Week 8
of an introductory programming course. Include doctest examples following
our Week 9 format. No advanced exception handling or complex data structures."
```

doctest Issues:

```python
# Run this in any cell to test your functions
if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose=True)
```

GitHub Workflow:

```
# To tag your final version (use GitHub web interface):
# Go to your repo → Releases → Create new release → Tag: v1.0
```

Academic Integrity Reminder:

- All course references must be authentic (we can verify)
- Code complexity must match your previous coursework

- doctest examples should reflect Week 9 lab style
- AI interactions should show you constraining responses to course level
- Reflection must connect to your actual learning experience
- GitHub commit history should show progressive work, not single large commits