AI Assisted Coding End Lab Exam 20111.2025 AIB09 and AIB10

📘 SET 1

Q1 (10 Marks)

A healthcare startup wants to automate the generation of clinical note summaries. Using prompt engineering, create a prompt that converts long doctor notes into short structured summaries.

Task: Write a Python function using an AI model (mock function ok) that takes doctor notes as input and returns a summarized version.

Include: docstrings, comments, and at least 3 test cases.

Q2 (10 Marks)

A logistics company needs a custom Stack implementation to store parcel tracking history.

Task: Implement a Stack using AI-assisted code completion and update it to track the latest 10 operations only.

Include: docstrings, comments, and test cases.

📘 SET 2

Q1 (10 Marks)

Question 1 (10 Marks):

Scenario: You need to process a list of customer orders. Each order is a dictionary with order_id, amount, and status ('Pending', 'Completed', 'Cancelled'). Write a Python function filter_orders that takes a list of orders and a status, and returns a new list containing only orders with that status. Use AI assistance to generate the list comprehension and also create a function calculate_total that returns the total amount for a given list of orders.

Include docstrings, comments, and test cases.

Q2 (10 Marks)

Scenario: You are designing a simple database for a library. Using AI, generate the SQL code to create two tables: Books (with fields: book_id, title, author, isbn, publication_year) and Members (member_id, name, email, join_date). Also, write a SQL query to find all books published after 2010 by the author "J.K. Rowling".

Include docstrings, comments, and test cases.

📘 SET 3

Q1. AI-Based Code Completion (Scenario)

- Scenario: Implement the Bubble Sort algorithm in Python to sort a list of integers in ascending order. Use AI to generate the initial code. Then, use a few-shot prompting technique to instruct the AI to also modify the code to count and return the number of swaps performed during the sorting process.

  Include: docstrings, comments, and test cases.

Q2. Web Frontend (Scenario)

Scenario: You are building a contact form for a website. Use AI to generate the HTML and CSS for a form with fields for Name, Email, Subject (as a dropdown with options: 'General Inquiry', 'Support', 'Feedback'), and Message (as a text area). The form should have a submit button and basic styling (padding, borders, etc.).

Include: docstrings, comments, and test cases.

📘 SET 4

Question 1 (10 Marks):

Scenario: Create a Python function clean_email_data that takes a list of email addresses and returns a cleaned list. The cleaning process should: convert all characters to lowercase, remove any leading/trailing whitespace, and remove any duplicates. Use AI to help you write efficient data pre-processing code.

Include: docstrings, comments, and test cases.

Question 2 (10 Marks):

Scenario: You need to integrate with a public API for weather data. The endpoint is https://api.weather.example.com/current?city={city_name}. Write a Python function get_weather that takes a city name, makes a request to this API, and returns the temperature. Include comprehensive error handling for scenarios like a non-200 HTTP status code, connection timeouts, and JSON decoding errors.

Include: docstrings, comments, and test cases.

📘 SET 5

Question 1 (10 Marks):

Scenario: Implement a Stack data structure in Python using a list. The class should have push, pop, and peek methods. Use AI code completion to generate the boilerplate code and then use a one-shot prompt to instruct the AI to also implement an is_empty method that returns True if the stack is empty.

Include: docstrings, comments, and test cases.

Question 2 (10 Marks):

Scenario: A file data.txt contains numbers separated by commas, but it's poorly formatted (e.g., "1, 2,3 ,4,,5"). Write a Python script that reads the file, uses AI-assisted data processing to parse the integers (ignoring empty strings and extra spaces), and prints the sum of these numbers.

Include: docstrings, comments, and test cases.

📘 SET 6

Question 1 (10 Marks):

Scenario: Write a Python function binary_search that implements the binary search algorithm on a sorted list. The function should return the index of the target element if found, otherwise -1. Use a zero-shot prompt to ask the AI to generate the code with clear, step-by-step comments explaining each part of the algorithm.

Include: docstrings, comments, and test cases.

Question 2 (10 Marks):

Scenario: You are building a login system and need to validate passwords. Write a Python function validate_password that checks if a password string is at least 8 characters long, contains at least one digit, one uppercase letter, and one special character (e.g., !, @, #, $). Use AI to generate the conditional checks and regular expressions.

Include: docstrings, comments, and test cases.

📘 SET 7

Q1. Data Processing (Scenario)

- Scenario: You have a CSV file employees.csv with columns Name, Department, Salary. Write a Python script that uses AI-assisted data processing to read the file, find the employee with the highest salary in each department, and print the results.
  (Include docs + comments + test cases.)

Q2: Real-Time Form Validation

Scenario: Build a user registration form with real-time validation for:

- Name (minimum 2 characters)
- Email (must contain @ and .)
- Password (minimum 8 characters with number and special character)
- Show green checkmarks when valid and red error messages when invalid.
- Use AI to generate the regular expressions and validation logic.

📘 SET 8

Q1. Web Frontend (Scenario)

Scenario: You're building a personal productivity app. Create a simple task manager with the following:

- An input field to add a new task.
- An "Add" button that appends the task to a list.
- Each task in the list should have a checkbox to mark it as complete (strikethrough text when checked) and a delete button to remove it.
- Use AI to generate the HTML structure, CSS for styling the completed tasks, and JavaScript for the interactive functionality.

Q2. Data Structures (Scenario)

Scenario: Implement a Queue data structure in Python. The class Queue should have enqueue, dequeue, and front methods. Use AI assistance to write the class. Then, using a few-shot prompt, instruct the AI to modify the queue to have a maximum size (a "bounded queue") and raise an exception if one tries to enqueue when the queue is full.

(Include docs + comments + test cases.)

📘 SET 9

Q1. Smart Agriculture Crop Monitoring System – Search Optimization

A smart agriculture company collects daily crop-health scores (0–100) from thousands of IoT sensors in a large field. The monitoring system needs to quickly search for sensors whose health score falls below a critical threshold (e.g., < 35). The engineer wants to speed up search performance using AI-assisted suggestion improvements to the algorithm.

Task:

1. Write Python code to implement an optimized search algorithm (binary search or any improved method) for finding all sensors below the threshold.
2. **Add docstrings, inline comments, and minimum 3 test cases.**
3. Print the list of sensor IDs whose readings are critical.

Question 2: E-Commerce Order Management

Scenario: "Quick Cart" needs a database to manage customers, products, and orders. Design a simplified schema with these tables:

customers (id, name, email, join_date)

products (id, name, price, category, stock_quantity)

orders (id, customer_id, order_date, status)

order_items (order_id, product_id, quantity, unit_price)

Tasks:

Write CREATE TABLE statements with primary keys, foreign keys, and constraints

Insert sample data for 2 customers, 3 products, and 2 orders

Write a query to find the total revenue generated from 'Electronics' category

Write a query to list customers who haven't placed any orders

📘 SET 10

Q1. Ride-Sharing App – Sorting & Route Optimization

A ride-sharing platform wants to assign drivers to customers based on minimum estimated time of arrival (ETA). The system receives live ETA estimates from multiple drivers and must sort these values quickly to choose the best driver. The developer decides to use AI-assisted sorting algorithm improvements to optimize performance, especially when the list size varies dynamically.

Task:

1. Implement a Python program that sorts driver ETAs using an AI-suggested sorting approach like merge sort and quick sort
2. **Include docstrings, inline comments, and minimum 3 test cases.**
3. Output the fastest available driver (lowest ETA) and the fully sorted ETA list.

Question 2: Library Management System

Scenario: "Community Library" needs to manage books, members, and loans. Design tables for:

books (book_id, title, author, isbn, publication_year, available_copies)

members (member_id, name, email, join_date, membership_type)

loans (loan_id, book_id, member_id, loan_date, due_date, return_date)

Tasks:

1. Create tables with proper constraints and data types
2. Insert sample data showing available and borrowed books
3. Write a query to find currently overdue books (not returned past due date)
4. Write a query to calculate the most popular author by loan count

SET-4:

Task-1:

Code:

Task1.py > clean_email_data

```python
def clean_email_data(email_list):
    """
    Clean a list of email addresses by:
    1. Converting all characters to lowercase
    2. Removing leading/trailing whitespace
    3. Removing duplicate email addresses while keeping order

    Parameters:
        email_list (list): A list containing email address strings.

    Returns:
        list: A cleaned list of unique, standardized email addresses.
    """

    # Step 1: Apply strip() and lower() to every email using a list comprehension
    cleaned_list = [email.strip().lower() for email in email_list]

    # Step 2: Remove duplicates while preserving the original order
    # Using a set to track seen items (AI-suggested optimal approach for O(n) performance)
    unique_emails = []
    seen = set()

    for email in cleaned_list:
        if email not in seen:
            unique_emails.append(email)
            seen.add(email)

    return unique_emails
```

Testcases code:

```python
import unittest
from Task1 import clean_email_data
class TestCleanEmailData(unittest.TestCase):

    def test_normal_input(self):
        emails = [" Alice@example.com ", "BOB@EXAMPLE.COM", "alice@example.com"]
        result = clean_email_data(emails)
        print("\nTest 1 Output:", result)
        expected = ['alice@example.com', 'bob@example.com']
        self.assertEqual(result, expected)

    def test_already_clean_emails(self):
        emails = ["user1@gmail.com", "user2@gmail.com"]
        result = clean_email_data(emails)
        print("\nTest 2 Output:", result)
        expected = ['user1@gmail.com', 'user2@gmail.com']
        self.assertEqual(result, expected)

    def test_extra_spaces_and_duplicates(self):
        emails = ["  TEST@MAIL.com", "test@mail.com  ", "Test@Mail.Com"]
        result = clean_email_data(emails)
        print("\nTest 3 Output:", result)
        expected = ['test@mail.com']
        self.assertEqual(result, expected)

    def test_empty_list(self):
        emails = []
        result = clean_email_data(emails)
        print("\nTest 4 Output:", result)
        expected = []
```

```
        expected = []
        self.assertEqual(result, expected)

    def test_mixed_case_uncommon_domains(self):
        emails = ["USER@YAHOO.CO.IN", "user@yahoo.co.in", "User@Gmail.Com"]
        result = clean_email_data(emails)
        print("\nTest 5 Output:", result)
        expected = ['user@yahoo.co.in', 'user@gmail.com']
        self.assertEqual(result, expected)


# Run tests and print outputs
if __name__ == "__main__":
    suite = unittest.defaultTestLoader.loadTestsFromTestCase(TestCleanEmailData)
    runner = unittest.TextTestRunner(verbosity=2)
    runner.run(suite)
```

Output:

```
ok
PS C:\Users\Namitha\OneDrive\Desktop\AIAC\End-Lab> & C:/Users/Namitha/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Namitha/One
Drive/Desktop/AIAC/End-Lab/test_task1.py
test_already_clean_emails (__main__.TestCleanEmailData.test_already_clean_emails) ...
Test 2 Output: ['user1@gmail.com', 'user2@gmail.com']
ok
test_empty_list (__main__.TestCleanEmailData.test_empty_list) ...
Test 4 Output: []
ok
test_extra_spaces_and_duplicates (__main__.TestCleanEmailData.test_extra_spaces_and_duplicates) ...
Test 3 Output: ['test@mail.com']
ok
test_mixed_case_uncommon_domains (__main__.TestCleanEmailData.test_mixed_case_uncommon_domains) ...
Test 5 Output: ['user@yahoo.co.in', 'user@gmail.com']
ok
test_normal_input (__main__.TestCleanEmailData.test_normal_input) ...
Test 1 Output: ['alice@example.com', 'bob@example.com']
ok

----------------------------------------------------------------------
Ran 5 tests in 0.006s

OK
PS C:\Users\Namitha\OneDrive\Desktop\AIAC\End-Lab>
```

Task-3:

Code:

```python
import requests
from unittest.mock import patch, MagicMock


def get_weather(city_name):
    """
    Fetch the current temperature for a given city using a public weather API.

    API Endpoint:
        https://api.weather.example.com/current?city={city_name}

    Handles:
    - Non-200 HTTP status codes
    - Connection timeouts
    - Connection errors
    - JSON decoding errors
    - Missing temperature field

    Parameters:
        city_name (str): Name of the city.

    Returns:
        float or None: Temperature if successful, otherwise None.
    """

    url = f"https://api.weather.example.com/current?city={city_name}"

    try:
        response = requests.get(url, timeout=5)

        # Handle non-200 responses
        if response.status_code != 200:
            print(f"Error: Received HTTP {response.status_code}")
            return None

        try:
            data = response.json()
        except ValueError:
            print("Error: Invalid JSON response")
            return None

        # Check for temperature field
        if "temperature" not in data:
            print("Error: 'temperature' field missing in response")
            return None

        return data["temperature"]

    except requests.Timeout:
        print("Error: Request timed out")
        return None

    except requests.ConnectionError:
        print("Error: Connection error occurred (check network)")
        return None

    except Exception as e:
        print(f"Unexpected error: {e}")
        return None
```

Test cases code:

```python
import unittest
from unittest.mock import patch, MagicMock
import requests
from Task2 import get_weather

class TestGetWeather(unittest.TestCase):

    @patch('Task2.requests.get')
    def test_successful_weather_fetch(self, mock_get):
        mock_response = MagicMock()
        mock_response.status_code = 200
        mock_response.json.return_value = {"temperature": 25.5}
        mock_get.return_value = mock_response

        result = get_weat
        self.assertEqual(result, 25.5)

    @patch('Task2.requests.get')
    def test_non_200_status_code(self, mock_get):
        mock_response = MagicMock()
        mock_response.status_code = 404
        mock_get.return_value = mock_response

        result = get_weather("InvalidCity")
        self.assertIsNone(result)

    @patch('Task2.requests.get')
    def test_invalid_json_response(self, mock_get):
        mock_response = MagicMock()
        mock_response.status_code = 200
        mock_response.json.side_effect = ValueError("Invalid JSON")
        mock_get.return_value = mock_response

        result = get_weather("Paris")
        self.assertIsNone(result)
```

(variable) result: Any | None

```python
    @patch('Task2.requests.get')
    def test_missing_temperature_field(self, mock_get):
        mock_response = MagicMock()
        mock_response.status_code = 200
        mock_response.json.return_value = {"humidity": 65}
        mock_get.return_value = mock_response

        result = get_weather("Tokyo")
        self.assertIsNone(result)

    @patch('Task2.requests.get')
    def test_timeout_error(self, mock_get):
        mock_get.side_effect = requests.Timeout()

        result = get_weather("Berlin")
        self.assertIsNone(result)

    @patch('Task2.requests.get')
    def test_connection_error(self, mock_get):
        mock_get.side_effect = requests.ConnectionError()

        result = get_weather("Madrid")
        self.assertIsNone(result)

    @patch('Task2.requests.get')
    def test_unexpected_error(self, mock_get):
        mock_get.side_effect = Exception("Unexpected error")

        result = get_weather("Rome")
        self.assertIsNone(result)


if __name__ == '__main__':
    unittest.main()
```

Output:

```
OK
PS C:\Users\Namitha\OneDrive\Desktop\AIAC\End-Lab> & C:/Users/Namitha/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Namitha/O
Drive/Desktop/AIAC/End-Lab/test_Task2.py
Error: Connection error occurred (check network)
.Error: Invalid JSON response
.Error: 'temperature' field missing in response
.Error: Received HTTP 404
..Error: Request timed out
.Unexpected error: Unexpected error
.
----------------------------------------------------------------------
Ran 7 tests in 0.005s

OK
PS C:\Users\Namitha\OneDrive\Desktop\AIAC\End-Lab>
```