| SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE | | DEPARTMENT OF COMPUTER SCIENCE ENGINEERING | |
|---|---|---|---|
| **Program Name:** B. Tech | | **Assignment Type: Lab** | **Program Name:** B. Tech |
| **Course Coordinator Name** | | Dr. Rishabh Mittal | |
| **Instructor(s) Name** | | Mr. S Naresh Kumar<br>Ms. B. Swathi<br>Dr. Sasanko Shekhar Gantayat<br>Mr. Md Sallauddin<br>Dr. Mathivanan<br>Mr. Y Srikanth<br>Ms. N Shilpa<br>Dr. Rishabh Mittal (Coordinator)<br>Dr. R. Prashant Kumar<br>Mr. Ankushavali MD<br>Mr. B Viswanath<br>Ms. Sujitha Reddy<br>Ms. A. Anitha<br>Ms. M.Madhuri<br>Ms. Katherashala Swetha<br>Ms. Velpula sumalatha<br>Mr. Bingi Raju | |
| **Course Code** | 23CS002PC304 | **Course Code** | 23CS002PC304 |
| **Year/Sem** | III/II | **Year/Sem** | III/II |
| **Date and Day of Assignment** | **Week5 – Wednesday** | **Date and Day of Assignment** | **Week5 – Wednesday** |
| **Duration** | 2 Hours | **Duration** | 2 Hours |
| **AssignmentNumber:** **10.3**(Present assignment number)/**24**(Total number of assignments) | | | |

| Q.No. | Question | *Expected Time to complete* |
|---|---|---|
| 1 | **Lab 9 – Code Review and Quality: Using AI to improve code quality and readability**<br><br>**Lab Objectives:**<br>• To apply AI-based prompt engineering for code review and quality improvement.<br>• To analyze code for readability, logic, performance, and | Week5 - Wednesday |

maintainability issues.
• To use Zero-shot, One-shot, and Few-shot prompting for improving code quality.
• To evaluate AI-generated improvements using standard coding practices.

**Lab Outcomes (LOs):** After completing this lab, students will be able to:

• Review and improve code quality using AI tools.
• Identify syntax, logic, and performance issues in code.
• Refactor code to improve readability and maintainability.
• Compare AI outputs generated using different prompting techniques.

**Problem Statement 1: AI-Assisted Bug Detection**

**Scenario:** A junior developer wrote the following Python function to calculate factorials:

```
def factorial(n):
    result = 1
    for i in range(1, n):
        result = result * i
    return result
```

**Instructions:**

1. Run the code and test it with factorial(5).
2. Use an AI assistant to:
    - Identify the logical bug in the code.
    - Explain why the bug occurs (e.g., off-by-one error).
    - Provide a corrected version.
3. Compare the AI's corrected code with your own manual fix.
4. Write a brief comparison: Did AI miss any edge cases (e.g., negative numbers, zero)?

**Expected Output:**

Corrected function should return 120 for factorial(5).

---

**Problem Statement 2: Task 2 — Improving Readability & Documentation**

**Scenario:**The following code works but is poorly written:

.

```
def calc(a, b, c):
    if c == "add":
        return a + b
    elif c == "sub":
        return a - b
```

```
    elif c == "mul":
        return a * b
    elif c == "div":
```

**Instructions:**

5. Use AI to:
    - Critique the function's readability, parameter naming, and lack of documentation.
    - Rewrite the function with:
        1. Descriptive function and parameter names.
        2. A complete docstring (description, parameters, return value, examples).
        3. Exception handling for division by zero.
        4. Consideration of input validation.
6. Compare the original and AI-improved versions.
7. Test both with valid and invalid inputs (e.g., division by zero, non-string operation).

**Expected Output:**

A well-documented, robust, and readable function that handles errors gracefully.

---

**Problem Statement 3: Enforcing Coding Standards**

**Scenario:** A team project requires PEP8 compliance. A developer submits:

```
def Checkprime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

**Instructions:**

8. Verify the function works correctly for sample inputs.
9. Use an AI tool (e.g., ChatGPT, GitHub Copilot, or a PEP8 linter with AI explanation) to:
    - List all PEP8 violations.
    - Refactor the code (function name, spacing, indentation, naming).
10. Apply the AI-suggested changes and verify functionality is preserved.
11. Write a short note on how automated AI reviews could streamline code reviews in large teams.

**Expected Output:**

A PEP8-compliant version of the function, e.g.:

```
def check_prime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

---

**Problem Statement 4: AI as a Code Reviewer in Real Projects**

**Scenario:**

In a GitHub project, a teammate submits:

```
def processData(d):
    return [x * 2 for x in d if x % 2 == 0]
```

**Instructions:**

1. Manually review the function for:
   - Readability and naming.
   - Reusability and modularity.
   - Edge cases (non-list input, empty list, non-integer elements).
2. Use AI to generate a code review covering:
   a. Better naming and function purpose clarity.
   b. Input validation and type hints.
   c. Suggestions for generalization (e.g., configurable multiplier).
3. Refactor the function based on AI feedback.
4. Write a short reflection on whether AI should be a standalone reviewer or an assistant.

**Expected Output:**

An improved function with type hints, validation, and clearer intent, e.g.:

```
from typing import List, Union

def double_even_numbers(numbers: List[Union[int, float]]) -> List[Union[int, float]]:
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")
    return [num * 2 for num in numbers if isinstance(num, (int, float)) and num % 2 == 0]
```

**Problem Statement 5: — AI-Assisted Performance Optimization**

**Scenario:** You are given a function that processes a list of integers, but it runs slowly on large datasets:

```
def sum_of_squares(numbers):
    total = 0
    for num in numbers:
        total += num ** 2
    return total
```

**Instructions:**

1. Test the function with a large list (e.g., range(1000000)).
2. Use AI to:
    - Analyze time complexity.
    - Suggest performance improvements (e.g., using built-in functions, vectorization with NumPy if applicable).
    - Provide an optimized version.
3. Compare execution time before and after optimization.
4. Discuss trade-offs between readability and performance.

**Expected Output:**

An optimized function, such as:

```
def sum_of_squares_optimized(numbers):
    return sum(x * x for x in numbers)
```