

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	Academic Year:2025-2026
Course Coordinator Name		Venkataramana Veeramsetty	
Instructor(s) Name		Dr. V. Venkataramana (Co-ordinator)	
		Dr. T. Sampath Kumar	
		Dr. Pramoda Patro	
		Dr. Brij Kishor Tiwari	
		Dr.J.Ravichander	
		Dr. Mohammand Ali Shaik	
		Dr. Anirodh Kumar	
		Mr. S.Naresh Kumar	
		Dr. RAJESH VELPULA	
		Mr. Kundhan Kumar	
		Ms. Ch.Rajitha	
		Mr. M Prakash	
		Mr. B.Raju	
		Intern 1 (Dharma teja)	
		Intern 2 (Sai Prasad)	
		Intern 3 (Sowmya)	
		NS_2 (Mounika)	
Course Code	24CS002PC215	Course Title	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week5 - Monday	Time(s)	
Duration	2 Hours	Applicable to Batches	
AssignmentNumber: 9.1(Present assignment number)/24(Total number of assignments)			

	Question	Expected Time to complete
1	Lab 9 – Code Review and Quality: Using AI to improve code quality and readability Lab Objectives <ul style="list-style-type: none"> • Inline comments • Docstrings 	Week 5 - Monday

- Auto-documentation tools
- AI-assisted summarization

Task Description #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
```

- Run the code and test it with factorial(5) (expected output = 120).
- Use AI (prompting) to review this code and identify the bug.
- Ask AI to suggest corrections and rewrite the code.
- Compare AI's corrected code with your own fix.

PROMPT:

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
```

Find error and modify the code.

```
def factorial(n):  
    result = 1  
    for i in range(1, n + 1): # The loop should go up to n (inclusive)  
        result = result * i  
    return result  
  
# Example usage:  
print(factorial(5)) # Output: 120
```

120

ORIGINAL CODE:

```
def factorial(n):  
    result = 1  
    for i in range(1, n): # The loop goes up to n (exclusive)  
        result = result * i  
    return result
```

CORRECTED CODE:

```
def factorial(n):  
    result = 1  
    for i in range(1, n + 1): # The loop goes up to n (inclusive)  
        result = result * i  
    return result
```

COMPARISON OF BOTH CODES:

The difference is in the `range()` function within the for loop.

- In your original code, `range(1, n)` generates numbers from 1 up to (but not including) `n`. This means for `factorial(5)`, the loop would iterate with `i` as 1, 2, 3, and 4. The result would be $1 * 1 * 2 * 3 * 4 = 24$.
- In the corrected code, `range(1, n + 1)` generates numbers from 1 up to and including `n`. For `factorial(5)`, the loop iterates with `i` as 1, 2, 3, 4, and 5. The result is $1 * 1 * 2 * 3 * 4 * 5 = 120$, which is the correct factorial of 5.

The correction ensures that the multiplication includes the number `n` itself, which is necessary for calculating the correct factorial.

OBSERVATION:

This code defines a function called `factorial` that calculates the factorial of a given non-negative integer `n`. It then calls this function with the input 5 and prints the result, which is 120.

Task Description #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":
```

```
        return a/b
```

- Use AI to review this code for readability, naming, and documentation issues.
- Prompt AI to rewrite the code with:
 - Clear function & variable names.
 - Proper docstrings.
 - Exception handling for division by zero.
- Compare the before-and-after versions to evaluate AI's contribution.

PROMPT:

```
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":
```

```
        return a/b
```

review this code for readability, naming, and documentation issues and also rewrite the code with

- Clear function & variable names.
- Proper docstrings.
- Exception handling for division by zero.

```

def perform_calculation(num1, num2, operation):
    """Performs basic arithmetic operations based on the specified operation.

    Args:
        num1: The first number.
        num2: The second number.
        operation: The operation to perform ('add', 'sub', 'mul', 'div').

    Returns:
        The result of the calculation.

    Raises:
        ValueError: If the operation is not one of the supported operations.
        ZeroDivisionError: If the operation is 'div' and num2 is zero.
    """
    if operation == "add":
        return num1 + num2
    elif operation == "sub":
        return num1 - num2
    elif operation == "mul":
        return num1 * num2
    elif operation == "div":
        if num2 == 0:
            raise ZeroDivisionError("Division by zero is not allowed.")
        return num1 / num2
    else:
        raise ValueError("Invalid operation. Supported operations are 'add', 'sub', 'mul', 'div'.")

```

COMPARING BEFORE AND AFTER CODES:

Original calc function:

- **Readability and Naming:**
 - Uses short, less descriptive variable names (a, b, c).
 - Lacks clear comments or explanations.
- **Documentation:**
 - No docstring to explain what the function does, its arguments, or what it returns.
- **Error Handling:**
 - No specific handling for division by zero, which would result in a ZeroDivisionError.

Refactored perform_calculation function:

- **Readability and Naming:**
 - Uses more descriptive variable names (number1, number2, operation).
 - The function name perform_calculation clearly indicates its purpose.
- **Documentation:**
 - Includes a comprehensive docstring explaining the function's purpose, arguments, and return value.
- **Error Handling:**
 - Explicitly checks if the divisor (number2) is zero for the "div" operation and returns an informative error message.

In summary, the refactored code is more readable, easier to understand, and more robust due to the improved naming conventions, documentation, and explicit error handling for division by zero.

OBSERVATION:

The code defines a well-documented function `perform_calculation` that handles basic arithmetic operations with error checking for division by zero and invalid inputs.

Task Description #3 (Enforcing Coding Standards)

Scenario: A team project requires following PEP8 style guide. One developer submits:

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

- Run this code and verify correctness.
- Use AI to perform a code quality review for PEP8 compliance.
- Prompt AI to return a refactored version with proper indentation, spacing, and naming conventions.
- Discuss how automated AI review can save time in large-scale projects.

PROMPT:

A team project requires following PEP8 style guide. One developer submits:

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

Run this code and verify correctness and perform a code quality review for PEP8 compliance and also to return a refactored version with proper indentation, spacing, and naming conventiontions.

```
def is_prime(n):
    """Checks if a number is prime."""
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

# Verify correctness with some test cases
print(f"Is 2 prime? {is_prime(2)}")
print(f"Is 4 prime? {is_prime(4)}")
print(f"Is 7 prime? {is_prime(7)}")
print(f"Is 9 prime? {is_prime(9)}")

# Code quality review (PEP8 compliance)
# The code already follows PEP8 guidelines for naming (is_prime), indentation, and spacing.
# A docstring is included to explain the function's purpose.
```

Discuss how automated AI review can save time in large-scale projects.

1. **Faster Feedback**
2. **Consistency**
3. **Early Bug Detection**
4. **Reduced Manual Effort**
5. **Scalability**
6. **Improved Code Quality.**
7. **Training and Onboarding**

OBSERVATION:

The code correctly checks for prime numbers with good code quality but could be more efficient for larger numbers.

Task Description #4 (AI as a Code Reviewer in Real Projects)

Scenario: You are part of a GitHub project. A teammate submits this pull request:

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

- Review this function manually for readability, reusability, and edge cases.
- Use AI to generate a code review comment, focusing on:
 - Naming conventions.
 - Input validation (e.g., what if d is not a list?).
 - Adding type hints.
- Modify the function based on AI's suggestions.
- Write a short reflection: *Would you trust AI as a standalone*

reviewer, or only as a support tool? Why?

PROMPT:

You are part of a GitHub project. A teammate submits this pull request:

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

Review this function manually for readability, reusability, and edge cases and generate a code review comment, focusing on:

- o Naming conventions.
- o Input validation (e.g., what if d is not a list?).
- o Adding type hints.

• Modify the function based on AI's suggestions.

```
from typing import List

def double_even_numbers(input_list: List[int]) -> List[int]:
    """
    Filters a list to keep only even numbers and doubles them.

    Args:
        input_list: A list of integers.

    Returns:
        A new list containing the doubled even numbers from the input list.

    Raises:
        TypeError: If the input is not a list or if any element is not an integer.
    """
    if not isinstance(input_list, list):
        raise TypeError("Input must be a list.")

    processed_data = []
    for x in input_list:
        if not isinstance(x, int):
            raise TypeError("All elements in the list must be integers.")
        if x % 2 == 0:
            processed_data.append(x * 2)
    return processed_data

# Example usage:
data = [1, 2, 3, 4, 5, 6]
result = double_even_numbers(data)
print(result)

# Example with invalid input:
# try:
#     double_even_numbers("not a list")
# except TypeError as e:
#     print(e)

# Example with non-integer element:
# try:
#     double_even_numbers([1, 2, 3.5, 4])
# except TypeError as e:
#     print(e)
```

[4, 8, 12]

Write a short reflection: *Would you trust AI as a standalone reviewer, or only as a support tool? Why?*

ANS:

- **Pattern Recognition and Efficiency:** AI is excellent at quickly scanning large amounts of code, identifying common patterns, suggesting improvements based on best practices, and catching potential errors like syntax issues or common anti-patterns. This significantly speeds up the initial review process.
- **Lack of Contextual Understanding:** While I can analyze code structure and suggest improvements based on general programming principles, I lack the deeper contextual understanding of the specific project, its overall architecture, business logic, and the nuances of the team's coding standards. A human reviewer can assess if the code aligns with the project's goals and fits seamlessly into the existing codebase.
- **Creativity and Design:** AI can't evaluate the creativity or elegance of a solution, or assess different design choices in the same way a human can. Human reviewers bring experience and intuition to evaluate the overall design and maintainability of the code.
- **Handling Ambiguity and Intent:** Sometimes, the intent behind a piece of code

might be unclear, or there might be multiple valid ways to approach a problem. Human reviewers can ask clarifying questions, discuss alternative solutions, and understand the developer's intent in a way that AI cannot.

- **Trust and Responsibility:** Ultimately, the responsibility for the quality and correctness of the code lies with the development team. AI can provide valuable suggestions, but the final decision-making and sign-off require human judgment and accountability

OBSERVATION:

This code defines a Python function `double_even_numbers` that takes a list of integers, filters out the odd numbers, doubles the remaining even numbers, and returns the result as a new list. It also includes input validation and type hints for better robustness and readability.