


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)			
Q.No.	Question		Expected Time to complete
1	<b>Lab 9 – Code Review and Quality: Using AI to improve code quality and readability</b>  <b>Lab Objectives</b> <ul style="list-style-type: none"> <li>• Inline comments</li> <li>• Docstrings</li> <li>• Auto-documentation tools</li> <li>• AI-assisted summarization</li> </ul>		Week5 - Monday

	<p><b>Task Description #1</b> (AI-Assisted Bug Detection)</p> <p><b>Scenario:</b> A junior developer wrote the following Python function to calculate factorials:</p> <pre>def factorial(n):     result = 1     for i in range(1, n):         result = result * i     return result</pre> <ul style="list-style-type: none"><li>• Run the code and test it with factorial(5) (expected output = 120).</li><li>• Use AI (prompting) to review this code and identify the bug.</li><li>• Ask AI to suggest corrections and rewrite the code.</li><li>• Compare AI’s corrected code with your own fix.</li><li>• <b>Prompt: <i>Review the following Python code for bugs and check if it gives the correct factorial output. Explain what is wrong and how to fix it.</i></b></li><li>• <b>def factorial(n):</b></li><li>• <b>    result = 1</b></li><li>• <b>    for i in range(1, n):</b></li><li>• <b>        result = result * i</b></li><li>• <b>    return result</b></li></ul> <p>Error: The original code for the factorial function had a small bug in the range of the for loop. It should have been range(1, n + 1) to include n in the multiplication. The corrected code is provided in the notebook and the output for factorial(5) is 120, which is correct.</p> <p><b>Prompt2: def factorial(n):</b></p> <pre>    result = 1     for i in range(1, n):         result = result * i     return result</pre> <p><b>Step 2 – Bug Fix &amp; Rewrite Prompt:</b></p> <p><b>Rewrite the above code so that it correctly calculates factorial(n), follows PEP8 style, and includes a docstring and inline comments. Add input validation for negative numbers and non-integers.</b></p>	
--	---	--

	<pre>def factorial(n):     """     Calculates the factorial of a non-negative integer.      Args:         n: An integer for which to calculate the factorial.      Returns:         The factorial of n.      Raises:         TypeError: If the input is not an integer.         ValueError: If the input is a negative integer.     """     # Input validation: Check if the input is an integer     if not isinstance(n, int):         raise TypeError("Input must be an integer.")      # Input validation: Check if the input is a negative number     if n &lt; 0:         raise ValueError("Factorial is not defined for negative numbers.")     # Base case: Factorial of 0 is 1     if n == 0:         return 1     # Calculate factorial using iteration     result = 1     for i in range(1, n + 1): # Fix the bug: include n in the range         result *= i # Use augmented assignment for conciseness     return result  # Test cases print(f"Factorial of 5: {factorial(5)}")</pre> <hr/> <p>Factorial of 5: 120</p>	
	<p><b>Task Description #2 (Improving Readability &amp; Documentation)</b></p> <p><b>Scenario:</b> The following code works but is poorly written:</p> <pre>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</pre> <ul style="list-style-type: none"> <li>• Use AI to review this code for readability, naming, and documentation issues.</li> <li>• Prompt AI to rewrite the code with:             <ul style="list-style-type: none"> <li>○ Clear function &amp; variable names.</li> </ul> </li> </ul>	

	<ul style="list-style-type: none"><li>○ Proper docstrings.</li><li>○ Exception handling for division by zero.</li><li>• Compare the before-and-after versions to evaluate AI's contribution.</li><li>• Prompt1: <i>Review this function for poor naming and readability issues. Suggest how to make it more understandable and user-friendly."</i></li><li>• <pre>def calc(a,b,c):</pre></li><li>• <pre>    if c=="add":</pre></li><li>• <pre>        return a+b</pre></li><li>• <pre>    elif c=="sub":</pre></li><li>• <pre>        return a-b</pre></li><li>• <pre>    elif c=="mul":</pre></li><li>• <pre>        return a*b</pre></li><li>• <pre>    elif c=="div":</pre></li><li>• <pre>        return a/b</pre></li><li>• </li></ul> <p>The original code has a few areas for improvement:</p> <ol style="list-style-type: none"><li>1. <b>Poor Naming:</b> The function name <code>calc</code> and the parameter names <code>a</code>, <code>b</code>, and <code>c</code> are not descriptive. It's unclear what the function does or what the parameters represent without looking at the code's logic.</li><li>2. <b>Lack of Readability:</b> The use of single-letter variable names makes the code harder to read and understand. The conditional statements using string comparisons are functional but could be made more robust.</li><li>3. <b>No Error Handling:</b> The function doesn't handle cases where an invalid operation is provided or when division by zero occurs.</li></ol> <p><b>Suggested Improvements:</b></p> <p>To make the function more understandable and user-friendly, we can:</p> <ol style="list-style-type: none"><li>1. <b>Use Descriptive Names:</b> Rename the function to reflect its purpose (e.g., <code>perform_operation</code>) and use clear parameter names (e.g., <code>num1</code>, <code>num2</code>, <code>operation</code>).</li><li>2. <b>Improve Readability:</b> Use full words for variable names. Consider using a dictionary to map operation strings to actual functions or a more explicit way to handle the operations.</li><li>3. <b>Add Error Handling:</b> Include checks for invalid operations and handle potential division by zero errors.</li></ol> <p><b>Prompt2: Rewrite the function using clear variable names, a proper function name, and a docstring. Add exception handling for division</b></p>	
--	---	--

	<p><b>by zero and make sure it follows PEP8 coding style.</b></p> <pre>def perform_operation(num1, num2, operation):     """     Performs a basic arithmetic operation on two numbers.      Args:         num1: The first number.         num2: The second number.         operation: A string representing the operation ("add", "sub", "mul", "div").      Returns:         The result of the operation.      Raises:         ValueError: If an invalid operation is provided or if division by zero occurs.         TypeError: If inputs are not numbers.     """     # Input validation: Check if inputs are numbers     if not all(isinstance(i, (int, float)) for i in [num1, num2]):         raise TypeError("Inputs must be numbers.")      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 ValueError("Division by zero is not allowed.")         return num1 / num2     else:         raise ValueError("Invalid operation. Please use 'add', 'sub', 'mul', or 'div'.")      # Test cases     print(f"10 + 5 = {perform_operation(10, 5, 'add')}")     print(f"10 - 5 = {perform_operation(10, 5, 'sub')}")     print(f"10 * 5 = {perform_operation(10, 5, 'mul')}")     print(f"10 / 5 = {perform_operation(10, 5, 'div')}")     # print(f"10 / 0 = {perform_operation(10, 0, 'div')}") # Uncomment to test division by zero     # print(f"10 % 5 = {perform_operation(10, 5, 'mod')}") # Uncomment to test invalid operation     # print(f"'a' + 5 = {perform_operation('a', 5, 'add')}") # Uncomment to test invalid input type</pre> <div><pre>10 + 5 = 15 10 - 5 = 5 10 * 5 = 50 10 / 5 = 2.0</pre></div>	
	<p><b>Task Description #3 (Enforcing Coding Standards)</b></p> <p><b>Scenario:</b> A team project requires following PEP8 style guide. One developer submits:</p> <pre>def Checkprime(n):     for i in range(2,n):         if n%i==0:             return False     return True</pre> <ul style="list-style-type: none"><li>• Run this code and verify correctness.</li><li>• Use AI to perform a code quality review for PEP8 compliance.</li><li>• Prompt AI to return a refactored version with proper indentation,</li></ul>	

	<p>spacing, and naming conventions.</p> <ul style="list-style-type: none"> <li>• Discuss how automated AI review can save time in large-scale projects.</li> <li>• <b>Prompt1: Run and test this function for correctness by checking a few values (e.g., prime and non-prime numbers) and confirm whether it works.</b></li> <li>• <b>def Checkprime(n):</b></li> <li>• <b>for i in range(2,n):</b></li> <li>• <b>if n%i==0:</b></li> <li>• <b>return False</b></li> <li>• <b>return True</b></li> </ul> <p>The function appears to work correctly for numbers greater than 1, but it incorrectly identifies 1 as a prime number. A prime number is defined as a natural number greater than 1 that has no positive divisors other than 1 and itself.</p> <ul style="list-style-type: none"> <li>• <b>Prompt2: Refactor the above code to be PEP8 compliant with proper indentation, spacing, descriptive function and variable names, inline comments, and a docstring explaining what it does.</b></li> </ul>	
--	--	--

	<pre>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</pre> <hr/> <pre>def Checkprime(n):     for i in range(2,n):         if n%i==0:             return False     return True</pre> <hr/> <pre># Test cases prime_numbers = [2, 3, 5, 7, 11, 13] non_prime_numbers = [1, 4, 6, 8, 9, 10, 12]  print("Testing with prime numbers:") for number in prime_numbers:     print(f"Is {number} prime? {Checkprime(number)}")  print("\nTesting with non-prime numbers:") for number in non_prime_numbers:     print(f"Is {number} prime? {Checkprime(number)}")</pre> <hr/> <p>⇒ Testing with prime numbers:</p> <pre>Is 2 prime? True Is 3 prime? True Is 5 prime? True Is 7 prime? True Is 11 prime? True Is 13 prime? True  Testing with non-prime numbers: Is 1 prime? True Is 4 prime? False Is 6 prime? False Is 8 prime? False Is 9 prime? False Is 10 prime? False Is 12 prime? False</pre> <p><b>Automated AI code reviews save time by quickly identifying common errors and style issues, allowing human reviewers to focus on complex logic. This improves code quality by enforcing standards and catching bugs early in the development cycle, especially in large-scale projects with many contributors.</b></p>	
	<b>Task Description #4</b> (AI as a Code Reviewer in Real Projects)	

	<p><b>Scenario:</b> You are part of a GitHub project. A teammate submits this pull request:</p> <pre>def processData(d):     return [x*2 for x in d if x%2==0]</pre> <ul style="list-style-type: none"> <li>• Review this function manually for readability, reusability, and edge cases.</li> <li>• Use AI to generate a code review comment, focusing on:             <ul style="list-style-type: none"> <li>○ Naming conventions.</li> <li>○ Input validation (e.g., what if d is not a list?).</li> <li>○ Adding type hints.</li> </ul> </li> <li>• Modify the function based on AI's suggestions.</li> <li>• Write a short reflection: <i>Would you trust AI as a standalone reviewer, or only as a support tool? Why?</i></li> <li>• <b>Prompt1: Manually review this function for readability, naming, and edge cases (e.g., invalid input). List at least two possible improvements."</b></li> <li>• <b>def processData(d):</b></li> <li>• <b>return [x*2 for x in d if x%2==0]</b></li> </ul> <p>The processData function takes an input d and returns a new list containing each even number from d multiplied by 2.</p> <p>Readability: The code uses a list comprehension, which is concise but might be less readable for beginners. The variable names d and x are not very descriptive.</p> <p>Naming: The function name processData is quite generic and doesn't clearly indicate what kind of processing is being done or what data is being processed.</p> <p>Edge Cases:</p> <p>Invalid Input Type: The code does not handle cases where the input d is not an iterable (e.g., an integer, a string that is not a sequence of numbers). It would raise a TypeError.</p> <p>Non-numeric Elements: If the input iterable d contains elements that are not numbers (e.g., strings), the code might raise a TypeError when trying to perform the modulo (%) or multiplication (*) operations.</p> <p>Empty Input: If the input d is an empty list or other empty iterable, the function will correctly return an empty list, which is expected behavior.</p> <p>Possible Improvements:</p> <p>Here are at least two possible improvements for the function:</p>	
--	--	--



	<p>Improve Naming and Readability: Use more descriptive names for the function and variables. For example, double_even_numbers(number_list). Add a docstring to explain what the function does, its arguments, and what it returns.</p> <p>Add Input Validation and Error Handling: Include checks to ensure the input d is an iterable and that its elements are numbers. This can prevent unexpected errors and make the function more robust. You could raise TypeError with a clear message if the input type or element types are incorrect.</p> <p><b>Prompt2: rewrite the function using your suggestions and add a descriptive docstring.</b></p> <pre>def process_and_double_even_numbers(data_list):     """     Filters a list to include only even numbers and doubles them.      Args:         data_list: A list or iterable containing numbers.      Returns:         A new list with even numbers from the input, doubled.      Raises:         TypeError: If the input is not an iterable or contains non-numeric elements.     """     # Input validation: Check if the input is an iterable     if not hasattr(data_list, '__iter__'):         raise TypeError("Input must be an iterable.")      processed_data = []     for item in data_list:         # Input validation: Check if elements are numbers         if not isinstance(item, (int, float)):             raise TypeError("Input iterable must contain only numbers.")         if item % 2 == 0:             processed_data.append(item * 2)     return processed_data  # Test cases print(process_and_double_even_numbers([1, 2, 3, 4, 5, 6])) print(process_and_double_even_numbers([10, 25, 30, 45, 50])) print(process_and_double_even_numbers([])) # print(process_and_double_even_numbers([1, 2, 'a', 4])) # Uncomment to test non-numeric element # print(process_and_double_even_numbers(123)) # Uncomment to test non-iterable input</pre> <pre>[4, 8, 12] [20, 60, 100] []</pre>	
--	--	--

I believe AI is currently best used as a support tool for code reviews. While AI can quickly catch common errors, style violations, and even suggest optimizations, it lacks the nuanced understanding of project context, business logic, and potential architectural implications that a human reviewer provides. Human oversight is crucial for complex decisions and ensuring the overall quality and maintainability of the codebase.

--	--	--