|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 - Thursday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | |  | | | |
| **AssignmentNumber:10.4**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | **Lab 10 – Code Review and Quality: Using AI to Improve Code Quality and Readability**  **Lab Objectives**   * Use AI for automated code review and quality enhancement. * Identify and fix syntax, logical, performance, and security issues in Python code. * Improve readability and maintainability through structured refactoring and comments. * Apply prompt engineering for targeted improvements. * Evaluate AI-generated suggestions against PEP 8 standards and software engineering best practices   **Task 1: Syntax and Error Detection**  **Task:** Identify and fix syntax, indentation, and variable errors in the given script.  # buggy\_code\_task1.py  def add\_numbers(a, b)  result = a + b  return reslt  print(add\_numbers(10 20))  **Expected Output**:   * Corrected code with proper syntax (: after function, fixed variable name, corrected function call). * AI should explain what was fixed.   # buggy\_code\_task1.py  def add\_numbers(a, b): # Added missing colon      result = a + b      return result # Corrected typo from 'reslt' to 'result'  print(add\_numbers(10, 20)) # Added missing comma  **Task 2: Logical and Performance Issue Review**  **Task**: Optimize inefficient logic while keeping the result correct.  # buggy\_code\_task2.py  def find\_duplicates(nums):  duplicates = []  for i in range(len(nums)):  for j in range(len(nums)):  if i != j and nums[i] == nums[j] and nums[i] not in duplicates:  duplicates.append(nums[i])  return duplicates  numbers = [1,2,3,2,4,5,1,6,1,2]  print(find\_duplicates(numbers))  **Expected Output**:   * More efficient duplicate detection (e.g., using sets). * AI should explain the optimization.   # buggy\_code\_task2.py  def find\_duplicates(nums):      num\_counts = {}      duplicates = []      for num in nums:          if num in num\_counts:              num\_counts[num] += 1              if num\_counts[num] == 2: # Add to duplicates list only the first time a duplicate is found                  duplicates.append(num)          else:              num\_counts[num] = 1      return duplicates  numbers = [1,2,3,2,4,5,1,6,1,2]  print(find\_duplicates(numbers))  O/P: [2,1]  **Task 3: Code Refactoring for Readability**  **Task**: Refactor messy code into clean, PEP 8–compliant, well-structured code.  # buggy\_code\_task3.py  def c(n):  x=1  for i in range(1,n+1):  x=x\*i  return x  print(c(5))  Expected Output:  Function renamed to calculate\_factorial.  Proper indentation, variable naming, docstrings, and formatting.  AI should provide a more readable version.  # buggy\_code\_task3.py  def calculate\_factorial(n):    """    Calculates the factorial of a non-negative integer.  Args:      n: A non-negative integer.    Returns:      The factorial of n.    """    factorial = 1    for i in range(1, n + 1):      factorial = factorial \* i    return factorial  # Example usage:  print(calculate\_factorial(5))  **Task 4: Security and Error Handling Enhancement**  **Task:** Add security practices and exception handling to the code.  # buggy\_code\_task4.py  import sqlite3  def get\_user\_data(user\_id):  conn = sqlite3.connect("users.db")  cursor = conn.cursor()  query = f"SELECT \* FROM users WHERE id = {user\_id};" # Potential SQL injection risk  cursor.execute(query)  result = cursor.fetchall()  conn.close()  return result  user\_input = input("Enter user ID: ")  print(get\_user\_data(user\_input))  **Expected Output:**  Safe query using parameterized SQL (? placeholders).  Try-except block for database errors.  Input validation before query execution.  # buggy\_code\_task4.py  import sqlite3  def get\_user\_data(user\_id):      """      Retrieves user data from the database based on user ID.    Args:          user\_id: The ID of the user to retrieve data for.      Returns:          A list of tuples containing user data, or None if an error occurs.      """      conn = None  # Initialize connection to None      try:          conn = sqlite3.connect("users.db")          cursor = conn.cursor()          # Use parameterized query to prevent SQL injection          query = "SELECT \* FROM users WHERE id = ?;"          cursor.execute(query, (user\_id,))          result = cursor.fetchall()          return result      except sqlite3.Error as e:          print(f"Database error: {e}")          return None      except ValueError:          print("Invalid input: User ID must be an integer.")          return None      finally:          if conn:              conn.close()  # Example usage:  user\_input = input("Enter user ID: ")  # Basic input validation  try:      user\_id = int(user\_input)      user\_data = get\_user\_data(user\_id)      if user\_data:          print("User Data:")          for row in user\_data:              print(row)  except ValueError:      print("Invalid input: User ID must be an integer.")  **Task 5: Automated Code Review Report Generation**  **Task**: Generate a **review report** for this messy code.  # buggy\_code\_task5.py  def calc(x,y,z):  if z=="add":  return x+y  elif z=="sub": return x-y  elif z=="mul":  return x\*y  elif z=="div":  return x/y  else: print("wrong")    print(calc(10,5,"add"))  print(calc(10,0,"div"))  **Expected Output**:  AI-generated **review report** should mention:   * + Missing docstrings   + Inconsistent formatting (indentation, inline return)   + Missing error handling for division by zero   + Non-descriptive function/variable names   + Suggestions for readability and PEP 8 compliance   def calculator(x, y, operation):      """      Performs basic arithmetic operations.      Args:        x: The first number.        y: The second number.        operation: A string indicating the operation ("add", "sub", "mul", "div").    Returns:        The result of the operation or an error message for invalid operations or division by zero.      """      if operation == "add":          return x + y      elif operation == "sub":          return x - y      elif operation == "mul":          return x \* y      elif operation == "div":          if y == 0:              return "Error: Division by zero!"          return x / y      else:          return "Error: Invalid operation!"  # Example usage:  print(calculator(10, 5, "add"))  print(calculator(10, 0, "div"))  print(calculator(10, 5, "mul"))  print(calculator(10, 5, "sub"))  print(calculator(10, 5, "mod")) | | | | | | Week5 - Thursday |  |