|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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** | | | | Dr. Rishabh Mittal | | | | | |
| **Instructor(s) Name** | | | | |  | | --- | | Mr. S Naresh Kumar | | Ms. B. Swathi | | Dr. Sasanko Shekhar Gantayat | | Mr. Md Sallauddin | | Dr. Mathivanan | | Mr. Y Srikanth | | Ms. N Shilpa | | Dr. Rishabh Mittal (Coordinator) | | Dr. R. Prashant Kumar | | Mr. Ankushavali MD | | Mr. B Viswanath | | Ms. Sujitha Reddy | | Ms. A. Anitha | | Ms. M.Madhuri | | Ms. Katherashala Swetha | | Ms. Velpula sumalatha | | Mr. Bingi Raju | | | | | | |
| **CourseCode** | | | 23CS002PC304 | **Course Title** | | AI Assisted Coding | | | |
| **Year/Sem** | | | III/II | **Regulation** | | R23 | | | |
| **Date and Day**  **of Assignment** | | | **Week4 – Friday** | **Time(s)** | | 23CSBTB01 To 23CSBTB52 | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | | All batches | | | |
| **Assignment Number: 7.5**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | **Lab 7: Error Debugging with AI: Systematic approaches to finding and fixing bugs**  Lab Objectives:   * To identify and correct syntax, logic, and runtime errors in Python programs using AI tools. * To understand common programming bugs and AI-assisted debugging suggestions. * To evaluate how AI explains, detects, and fixes different types of coding errors. * To build confidence in using AI to perform structured debugging practices.   Lab Outcomes (LOs):  After completing this lab, students will be able to:   * Use AI tools to detect and correct syntax, logic, and runtime errors. * Interpret AI-suggested bug fixes and explanations. * Apply systematic debugging strategies supported by AI-generated insights.   Refactor buggy code using responsible and reliable programming patterns. | | | | | | Week4 -Monday |  |
|  |  | **Task 1 (Mutable Default Argument – Function Bug)**  Task: Analyze given code where a mutable default argument causes unexpected behavior. Use AI to fix it.  # Bug: Mutable default argument  def add\_item(item, items=[]):  items.append(item)  return items  print(add\_item(1))  print(add\_item(2))    Expected Output: Corrected function avoids shared list bug. | | | | | |  |  |
|  |  | **Task 2 (Floating-Point Precision Error)**  Task: Analyze given code where floating-point comparison fails. Use AI to correct with tolerance.  # Bug: Floating point precision issue  def check\_sum():  return (0.1 + 0.2) == 0.3  print(check\_sum())    Expected Output: Corrected function | | | | | |  |  |
|  |  | **Task 3 (Recursion Error – Missing Base Case)**  Task: Analyze given code where recursion runs infinitely due to missing base case. Use AI to fix.  # Bug: No base case  def countdown(n):  print(n)  return countdown(n-1)  countdown(5)    Expected Output : Correct recursion with stopping condition**.** | | | | | |  |  |
|  |  | **Task 4 (Dictionary Key Error)**  Task: Analyze given code where a missing dictionary key causes error. Use AI to fix it.  # Bug: Accessing non-existing key  def get\_value():  data = {"a": 1, "b": 2}  return data["c"]  print(get\_value())    Expected Output: Corrected with .get() or error handling. | | | | | |  |  |
|  |  | **Task 5 (Infinite Loop – Wrong Condition)**  Task: Analyze given code where loop never ends. Use AI to detect and fix it.  # Bug: Infinite loop  def loop\_example():  i = 0  while i < 5:  print(i)    Expected Output: Corrected loop increments i. | | | | | |  |  |
|  |  | **Task 6 (Unpacking Error – Wrong Variables)**  Task: Analyze given code where tuple unpacking fails. Use AI to fix it.  # Bug: Wrong unpacking  a, b = (1, 2, 3)    Expected Output: Correct unpacking or using \_ for extra values. | | | | | |  |  |
|  |  | **Task 7 (Mixed Indentation – Tabs vs Spaces)**  **Task:** Analyze given code where mixed indentation breaks execution. Use AI to fix it.  # Bug: Mixed indentation  def func():  x = 5  y = 10  return x+y    Expected Output : Consistent indentation applied. | | | | | |  |  |
|  |  | **Task 8 (Import Error – Wrong Module Usage)**  Task: Analyze given code with incorrect import. Use AI to fix.  # Bug: Wrong import  import maths  print(maths.sqrt(16))    Expected Output: Corrected to import math | | | | | |  |  |
|  |  | **Task 9 (Unreachable Code – Return Inside Loop)**  **Task:** Analyze given code where a return inside a loop prevents full iteration. Use AI to fix it.  # Bug: Early return inside loop  def total(numbers):  for n in numbers:  return n  print(total([1,2,3]))    **Expected Output:** Corrected code accumulates sum and returns after loop. | | | | | |  |  |
|  |  | **Task 10 (Name Error – Undefined Variable)**  Task: Analyze given code where a variable is used before being defined. Let AI detect and fix the error.  # Bug: Using undefined variable  def calculate\_area():  return length \* width  print(calculate\_area())    Requirements:   * Run the code to observe the error. * Ask AI to identify the missing variable definition. * Fix the bug by defining length and width as parameters. * Add 3 assert test cases for correctness.   Expected Output :   * Corrected code with parameters. * AI explanation of the bug.   Successful execution of assertions. | | | | | |  |  |
|  |  | **Task 11 (Type Error – Mixing Data Types Incorrectly)**  Task: Analyze given code where integers and strings are added incorrectly. Let AI detect and fix the error.  # Bug: Adding integer and string  def add\_values():  return 5 + "10"  print(add\_values())    Requirements:   * Run the code to observe the error. * AI should explain why int + str is invalid. * Fix the code by type conversion (e.g., int("10") or str(5)). * Verify with 3 assert cases.   Expected Output #6:   * Corrected code with type handling. * AI explanation of the fix.   Successful test validation. | | | | | |  |  |
|  |  | **Task 12 (Type Error – String + List Concatenation)**  Task: Analyze code where a string is incorrectly added to a list.  # Bug: Adding string and list  def combine():  return "Numbers: " + [1, 2, 3]  print(combine())    Requirements:   * Run the code to observe the error. * Explain why str + list is invalid. * Fix using conversion (str([1,2,3]) or " ".join()). * Verify with 3 assert cases.   Expected Output:   * Corrected code * Explanation * Successful test validation   **Task 13 (Type Error – Multiplying String by Float)**  Task: Detect and fix code where a string is multiplied by a float.  # Bug: Multiplying string by float  def repeat\_text():  return "Hello" \* 2.5  print(repeat\_text())    Requirements:   * Observe the error. * Explain why float multiplication is invalid for strings. * Fix by converting float to int. * Add 3 assert test cases.   **Task 14 (Type Error – Adding None to Integer)**  Task: Analyze code where None is added to an integer.  # Bug: Adding None and integer  def compute():  value = None  return value + 10  print(compute())    Requirements:   * Run and identify the error. * Explain why NoneType cannot be added. * Fix by assigning a default value. * Validate using asserts.   **Task 15 (Type Error – Input Treated as String Instead of Number)**  Task: Fix code where user input is not converted properly.  # Bug: Input remains string  def sum\_two\_numbers():  a = input("Enter first number: ")  b = input("Enter second number: ")  return a + b  print(sum\_two\_numbers())    Requirements:   * Explain why input is always string. * Fix using int() conversion. * Verify with assert test cases. | | | | | |  |  |