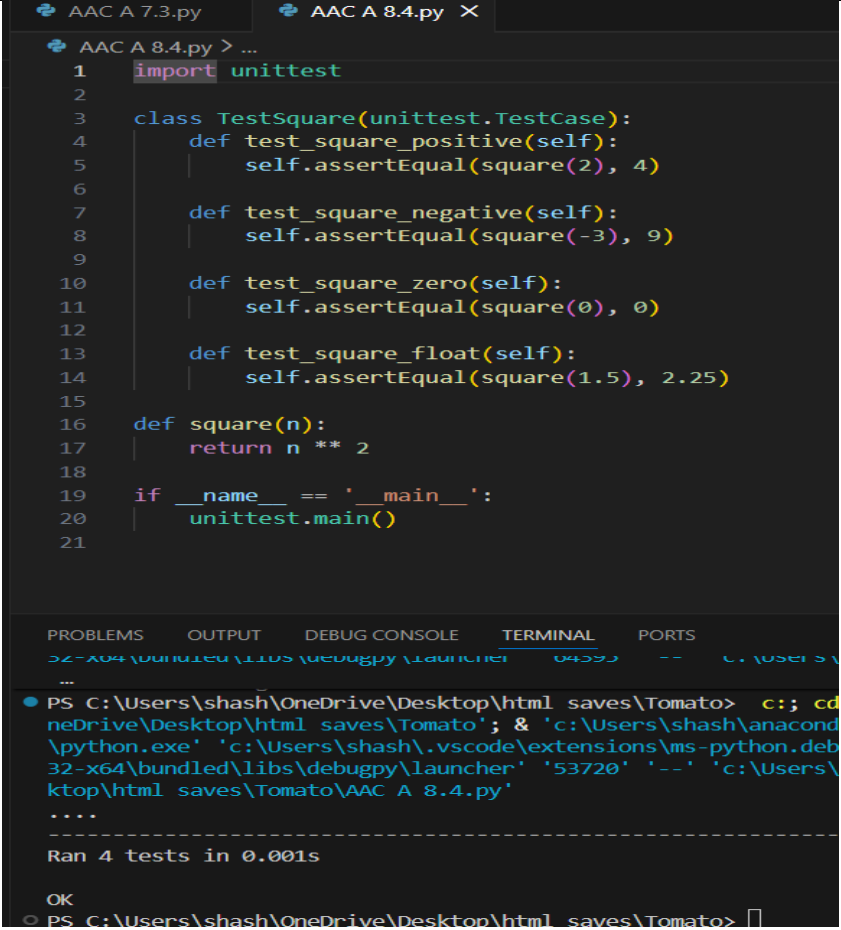


Name: B.NITHIIN H.No: 2303A51803 Batch: 26

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		<table border="1"> <tr><td>Mr. S Naresh Kumar</td><td></td></tr> <tr><td>Ms. B. Swathi</td><td></td></tr> <tr><td>Dr. Sasanko Shekhar Gantayat</td><td></td></tr> <tr><td>Mr. Md Sallauddin</td><td></td></tr> <tr><td>Dr. Mathivanan</td><td></td></tr> <tr><td>Mr. Y Srikanth</td><td></td></tr> <tr><td>Ms. N Shilpa</td><td></td></tr> <tr><td>Dr. Rishabh Mittal (Coordinator)</td><td></td></tr> <tr><td>Dr. R. Prashant Kumar</td><td></td></tr> <tr><td>Mr. Ankushavali MD</td><td></td></tr> <tr><td>Mr. B Viswanath</td><td></td></tr> <tr><td>Ms. Sujitha Reddy</td><td></td></tr> <tr><td>Ms. A. Anitha</td><td></td></tr> <tr><td>Ms. M. Madhuri</td><td></td></tr> <tr><td>Ms. Katherashala Swetha</td><td></td></tr> <tr><td>Ms. Velpula sumalatha</td><td></td></tr> <tr><td>Mr. Bingi Raju</td><td></td></tr> <tr><td>Mr. G. Kranthi</td><td></td></tr> </table>		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		Mr. G. Kranthi	
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																																							
Mr. G. Kranthi																																							
Course Code	23CS002PC304	Course Title	AI Assisted Coding																																				
Year/Sem	III/I	Regulation	R23																																				
Date and Day of Assignment	Week 4 - Thursday	Time(s)	23CSBTB01 To 23CSBTB52																																				
Duration	2 Hours	Applicable to Batches	All Batches																																				
Assignment Number: 8.4 (Present assignment number) / 24 (Total number of assignments)																																							
Q.No.	Question	Expected Time to complete																																					
1	Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases	Week 4																																					

	<p>Lab Objectives:</p> <ul style="list-style-type: none"> • To introduce students to test-driven development (TDD) using AI code generation tools. • To enable the generation of test cases before writing code implementations. • To reinforce the importance of testing, validation, and error handling. • To encourage writing clean and reliable code based on AI-generated test expectations. <p>Lab Outcomes (LOs):</p> <p>By the end of this lab, students will be able to:</p> <ul style="list-style-type: none"> • Apply TDD methodology using AI tools. • Generate test cases before writing the actual code logic. • Validate and refactor code based on test outcomes. • Use Python's unittest or pytest libraries for test-driven development. • Develop confidence in debugging and improving code with AI guidance. 	
	<p>Task 1: Developing a Utility Function Using TDD</p> <p>Scenario You are working on a small utility library for a larger software system. One of the required functions should calculate the square of a given number, and correctness is critical because other modules depend on it.</p> <p>Task Description Following the Test Driven Development (TDD) approach:</p> <ol style="list-style-type: none"> 1. First, write unit test cases to verify that a function correctly returns the square of a number for multiple inputs. 2. After defining the test cases, use GitHub Copilot or Cursor AI to generate the function implementation so that all tests pass. <p>Ensure that the function is written only after the tests are created.</p> <p>Expected Outcome</p> <ul style="list-style-type: none"> • A separate test file and implementation file • Clearly written test cases executed before implementation • AI-assisted function implementation that passes all tests • Demonstration of the TDD cycle: <i>test</i> → <i>fail</i> → <i>implement</i> → <i>pass</i> 	

	 <pre> AAC A 8.4.py > ... 1 import unittest 2 3 class TestSquare(unittest.TestCase): 4 def test_square_positive(self): 5 self.assertEqual(square(2), 4) 6 7 def test_square_negative(self): 8 self.assertEqual(square(-3), 9) 9 10 def test_square_zero(self): 11 self.assertEqual(square(0), 0) 12 13 def test_square_float(self): 14 self.assertEqual(square(1.5), 2.25) 15 16 def square(n): 17 return n ** 2 18 19 if __name__ == '__main__': 20 unittest.main() 21 </pre> <pre> PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato> c:; cd neDrive\Desktop\html saves\Tomato; & 'c:\Users\shash\anacond \python.exe' 'c:\Users\shash\.vscode\extensions\ms-python.deb 32-x64\bundled\libs\debugpy\launcher' '53720' '--' 'c:\Users\ ktop\html saves\Tomato\AAC A 8.4.py' Ran 4 tests in 0.001s OK PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato> </pre>	
	<p>Task 2: Email Validation for a User Registration System</p> <p>Scenario You are developing the backend of a user registration system. One requirement is to validate user email addresses before storing them in the database.</p> <p>Task Description Apply Test Driven Development by:</p> <ol style="list-style-type: none"> 1. Writing unit test cases that define valid and invalid email formats (e.g., missing @, missing domain, incorrect structure). 2. Using AI assistance to implement the validate_email() function based strictly on the behavior described by the test cases. <p>The implementation should be driven entirely by the test expectations.</p> <p>Expected Outcome</p> <ul style="list-style-type: none"> • Well-defined unit tests using unittest or pytest • An AI-generated email validation function • All test cases passing successfully • Clear alignment between test cases and function behavior 	

	 <pre> AAC A 8.4.py > ... 1 import unittest 2 import re 3 4 class TestEmailValidation(unittest.TestCase): 5 def test_valid_email(self): 6 self.assertTrue(validate_email("user@example.com")) 7 8 def test_email_without_at(self): 9 self.assertFalse(validate_email("userexample.com")) 10 11 def test_email_without_domain(self): 12 self.assertFalse(validate_email("user@")) 13 14 def test_email_with_invalid_chars(self): 15 self.assertFalse(validate_email("user@exam ple.com")) 16 17 def test_email_with_multiple_at(self): 18 self.assertFalse(validate_email("user@@example.com")) 19 20 def validate_email(email): 21 pattern = r'^[a-zA-Z0-9,_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$' 22 return re.match(pattern, email) is not None 23 24 if __name__ == '__main__': 25 unittest.main() </pre> <p>PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS</p> <p>PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato> c;; cd 'c:\Users\shash\OneDrive\Desktop\html saves\Tomato'; & 'c:\Users\shash\anaconda3\envs\Shashidh\python.exe' 'c:\Users\shash\.vscode\extensions\ms-python.debugpy-2025.18.0-w32-x64\bundled\libs\debugpy\launcher' '54475' '--' 'c:\Users\shash\OneDrive\Desktop\html saves\Tomato\AAC A 8.4.py'</p> <p>.....</p> <p>Ran 5 tests in 0.001s</p> <p>OK</p>	
	<p>Task 3: Decision Logic Development Using TDD</p> <p>Scenario In a grading or evaluation module, a function is required to determine the maximum value among three inputs. Accuracy is essential, as incorrect results could affect downstream decision logic.</p> <p>Task Description Using the TDD methodology:</p> <ol style="list-style-type: none"> 1. Write test cases that describe the expected output for different combinations of three numbers. 2. Prompt GitHub Copilot or Cursor AI to implement the function logic based on the written tests. <p>Avoid writing any logic before test cases are completed.</p> <p>Expected Outcome</p> <ul style="list-style-type: none"> • Comprehensive test cases covering normal and edge cases • AI-generated function implementation • Passing test results demonstrating correctness • Evidence that logic was derived from tests, not assumptions 	

	<div><div>AAC A 7.3.pyAAC A 8.4.py X</div><div><div>AAC A 8.4.py > TestMaxOfThree > test_max_first_max</div><pre>1 import unittest 2 3 class TestMaxOfThree(unittest.TestCase): 4 def test_max_all_positive(self): 5 self.assertEqual(max_of_three(1, 2, 3), 3) 6 7 def test_max_with_negatives(self): 8 self.assertEqual(max_of_three(-1, -2, -3), -1) 9 10 def test_max_mixed(self): 11 self.assertEqual(max_of_three(-1, 5, 0), 5) 12 13 def test_max_duplicates(self): 14 self.assertEqual(max_of_three(2, 2, 2), 2) 15 16 def test_max_first_max(self): 17 self.assertEqual(max_of_three(10, 5, 7), 10) 18 19 def max_of_three(a, b, c): 20 return max(a, b, c) 21 22 if __name__ == '__main__': 23 unittest.main()</pre></div><div><div>PROBLEMSOUTPUTDEBUG CONSOLETERMINALPORTS</div><div>PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato> python "AAC A 8.4.py"</div><div>Ran 5 tests in 0.001s</div><div>OK</div><div>PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato> python "AAC A 8.4.py"</div><div>.....</div><div>Ran 5 tests in 0.000s</div><div>OK</div></div></div>	
	<div><div>Task 4: Shopping Cart Development with AI-Assisted TDD</div><div>Scenario You are building a simple shopping cart module for an e-commerce application. The cart must support adding items, removing items, and calculating the total price accurately.</div><div>Task Description Follow a test-driven approach:<ol style="list-style-type: none">Write unit tests for each required behavior:<ul style="list-style-type: none">Adding an itemRemoving an itemCalculating the total priceAfter defining all tests, use AI tools to generate the ShoppingCart class and its methods so that the tests pass.</div><div>Focus on behavior-driven testing rather than implementation details.</div><div>Expected Outcome</div></div>	

- Unit tests defining expected shopping cart behavior
- AI-generated class implementation
- All tests passing successfully
- Clear demonstration of TDD applied to a class-based design

```

AAC A 7.3.py  AAC A 8.4.py X
AAC A 8.4.py > TestShoppingCart > test_calculate_total_empty
1  import unittest
2
3  class TestShoppingCart(unittest.TestCase):
4      def setUp(self):
5          self.cart = ShoppingCart()
6
7      def test_add_item(self):
8          self.cart.add_item(("apple", 1.0))
9          self.assertEqual(len(self.cart.items), 1)
10         self.assertEqual(self.cart.items[0], ("apple", 1.0))
11
12     def test_remove_item(self):
13         self.cart.add_item(("apple", 1.0))
14         self.cart.remove_item(("apple", 1.0))
15         self.assertEqual(len(self.cart.items), 0)
16
17     def test_calculate_total_empty(self):
18         self.assertEqual(self.cart.calculate_total(), 0.0)
19
20     def test_calculate_total_with_items(self):
21         self.cart.add_item(("apple", 1.0))
22         self.cart.add_item(("banana", 0.5))
23         self.assertEqual(self.cart.calculate_total(), 1.5)
24
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato> python "AAC A 8.4.py"
-----
Ran 5 tests in 0.000s
OK
PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato> python "AAC A 8.4.py"
....
-----
Ran 4 tests in 0.000s
OK
PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato>

```

	 <pre> AAC A 7.3.py AAC A 8.4.py X AAC A 8.4.py > TestShoppingCart > test_calculate_total_empty 3 class TestShoppingCart(unittest.TestCase): 20 def test_calculate_total_with_items(self): 21 self.cart.add_item(("apple", 1.0)) 22 self.cart.add_item(("banana", 0.5)) 23 self.assertEqual(self.cart.calculate_total(), 1.5) 24 25 class ShoppingCart: 26 def __init__(self): 27 self.items = [] 28 29 def add_item(self, item): 30 self.items.append(item) 31 32 def remove_item(self, item): 33 if item in self.items: 34 self.items.remove(item) 35 36 def calculate_total(self): 37 return sum(price for _, price in self.items) 38 39 if __name__ == '__main__': 40 unittest.main() 41 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato> python "AAC A 8.4.py" ----- Ran 5 tests in 0.000s OK PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato> python "AAC A 7.3.py" ----- Ran 4 tests in 0.000s OK </pre>	
	<p>Task 5: String Validation Module Using TDD</p> <p>Scenario You are working on a text-processing module where a function is required to identify whether a given string is a palindrome. The function must handle different cases and inputs reliably.</p> <p>Task Description Using Test Driven Development:</p> <ol style="list-style-type: none"> Write test cases for a palindrome checker covering: <ul style="list-style-type: none"> Simple palindromes Non-palindromes Case variations Use GitHub Copilot or Cursor AI to generate the <code>is_palindrome()</code> function based on the test case expectations. <p>The function should be implemented only after tests are written.</p> <p>Expected Outcome</p> <ul style="list-style-type: none"> Clearly written test cases defining expected behavior 	

- AI-assisted implementation of the palindrome checker
- All test cases passing successfully
- Evidence of TDD methodology applied correctly

```

AAC A 7.3.py AAC A 8.4.py
AAC A 8.4.py > ...
1  import unittest
2  class TestPalindrome(unittest.TestCase):
3      def test_simple_palindrome(self):
4          self.assertTrue(is_palindrome("radar"))
5
6      def test_non_palindrome(self):
7          self.assertFalse(is_palindrome("hello"))
8
9      def test_case_insensitive(self):
10         self.assertTrue(is_palindrome("Racecar"))
11
12     def test_empty_string(self):
13         self.assertTrue(is_palindrome(""))
14
15     def test_single_character(self):
16         self.assertTrue(is_palindrome("a"))
17
18     def is_palindrome(s):
19         s = s.lower()
20         return s == s[::-1]
21
22     if __name__ == '__main__':
23         unittest.main()
24
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato> python "AAC A 8.4.py"
-----
Ran 4 tests in 0.000s

OK
PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato> python "AAC A 8.4.py"
.....
-----
Ran 5 tests in 0.000s

OK
PS C:\Users\shash\OneDrive\Desktop\html saves\Tomato>

```

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots