ROLL NO:2403A51185
BATCH NO:09

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName: B. Tech		**Assignment Type: Lab**	**AcademicYear:** 2025-2026
CourseCoordinatorName		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)	
CourseCode	24CS002PC215	**CourseTitle**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week4 - Tuesday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber: **8.2**(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*ExpectedTime to complete*
1	Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases **Lab Objectives:** • To introduce students to test-driven development (TDD) using AI code generation tools. • To enable the generation of test cases before writing code implementations.	Week4 - Wednesday

- To reinforce the importance of testing, validation, and error handling.
- To encourage writing clean and reliable code based on AI-generated test expectations.

Lab Outcomes (LOs):
After completing this lab, students will be able to:

- Use AI tools to write test cases for Python functions and classes.
- Implement functions based on test cases in a test-first development style.
- Use unittest or pytest to validate code correctness.
- Analyze the completeness and coverage of AI-generated tests.
- Compare AI-generated and manually written test cases for quality and logic

Task Description#1

Use AI to generate test cases for a function is_prime(n) and then implement the function.

Requirements:

- Only integers > 1 can be prime.
- Check edge cases: 0, 1, 2, negative numbers, and large primes.

Expected Output#1

- A working prime checker that passes AI-generated tests using edge coverage.

```
def is_prime(n):
    if not isinstance(n, int) or n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n ** 0.5) + 1, 2):
        if n % i == 0:
            return False
    return True

# AI-generated test cases for edge coverage
test_cases = [
    (None, False),          # Non-integer input: None
    ("7", False),           # Non-integer input: string
    ([], False),            # Non-integer input: list
    (0, False),             # Edge: zero
    (1, False),             # Edge: one
    (2, True),              # Edge: smallest prime
    (3, True),              # Smallest odd prime
    (-2, False),            # Negative even number
    (-3, False),            # Negative odd number
    (4, False),             # Smallest composite even
```

```
    (4, False),             # Smallest composite even
    (9, False),             # Odd composite (3*3)
    (25, False),            # Odd composite (5*5)
    (17, True),             # Small prime
    (18, False),            # Small composite
    (7919, True),           # Large prime
    (7920, False),          # Large composite
    (999983, True),         # Very large prime
    (1000000, False),       # Very large composite
    (2147483647, True),     # Large prime (2^31-1, Mersenne prime)
    (2147483646, False),    # Large composite (even, just below Mersenne prime)
]

for n, expected in test_cases:
    result = is_prime(n)
    print(f"is_prime({n}) = {result} (Expected: {expected})")
```

```
is_prime(None) = False (Expected: False)
is_prime(7) = False (Expected: False)
is_prime([]) = False (Expected: False)
is_prime(0) = False (Expected: False)
is_prime(1) = False (Expected: False)
is_prime(2) = True (Expected: True)
is_prime(3) = True (Expected: True)
is_prime(-2) = False (Expected: False)
is_prime(-3) = False (Expected: False)
is_prime(4) = False (Expected: False)
is_prime(9) = False (Expected: False)
is_prime(25) = False (Expected: False)
is_prime(17) = True (Expected: True)
is_prime(18) = False (Expected: False)
is_prime(7919) = True (Expected: True)
is_prime(18) = False (Expected: False)
is_prime(18) = False (Expected: False)
is_prime(18) = False (Expected: False)
is_prime(7919) = True (Expected: True)
is_prime(7920) = False (Expected: False)
is_prime(999983) = True (Expected: True)
is_prime(1000000) = False (Expected: False)
is_prime(2147483647) = True (Expected: True)
is_prime(2147483646) = False (Expected: False)
```

Task Description#2 (Loops)

- Ask AI to generate test cases for celsius_to_fahrenheit(c) and fahrenheit_to_celsius(f).

Requirements

- Validate known pairs: 0°C = 32°F, 100°C = 212°F.
- Include decimals and invalid inputs like strings or None

Expected Output#2

Dual conversion functions with complete test coverage and safe type handling

```python
def celsius_to_fahrenheit(c):
    if not isinstance(c, (int, float)):
        return None
    return c * 9 / 5 + 32

def fahrenheit_to_celsius(f):
    if not isinstance(f, (int, float)):
        return None
    return (f - 32) * 5 / 9

# Test cases for celsius_to_fahrenheit and fahrenheit_to_celsius
test_cases = [
    # (function, input, expected_output)
    (celsius_to_fahrenheit, 0, 32),
    (celsius_to_fahrenheit, 100, 212),
    (fahrenheit_to_celsius, 32, 0),
    (fahrenheit_to_celsius, 212, 100),
    (celsius_to_fahrenheit, 37, 98.6),
    (fahrenheit_to_celsius, 98.6, 37),
    (celsius_to_fahrenheit, 25.5, 77.9),
    (fahrenheit_to_celsius, 77.9, 25.5),
    (celsius_to_fahrenheit, "abc", None),
    (fahrenheit_to_celsius, None, None),
    (celsius_to_fahrenheit, None, None),
    (fahrenheit_to_celsius, "xyz", None),
]

for func, value, expected in test_cases:
    result = func(value)
    # For floats, allow a small margin of error
    if isinstance(expected, float):
        passed = abs(result - expected) < 0.01 if result is not None else False
    else:
        passed = result == expected
    print(f"{func.__name__}({value}) = {result} (Expected: {expected}) - {'PASS' if passed else 'FAIL'}")
```

```
celsius_to_fahrenheit(abc) = None (Expected: None) - PASS
fahrenheit_to_celsius(None) = None (Expected: None) - PASS
celsius_to_fahrenheit(None) = None (Expected: None) - PASS
fahrenheit_to_celsius(xyz) = None (Expected: None) - PASS
PS C:\Users\pooji\OneDrive\Desktop\btech\2-1>
```

Task Description#3

Use AI to write test cases for a function count_words(text) that returns the number of words in a sentence.

Requirement

Handle normal text, multiple spaces, punctuation, and empty strings.

Expected Output#3
Accurate word count with robust test case validation.

lab22.py > ...

```python
import re

def count_words(text):
    # Use regex to split by word boundaries, ignoring punctuation and multiple spaces
    words = re.findall(r'\b\w+\b', text)
    return len(words)

# Test cases for count_words
test_cases = [
    ("Hello world", 2),                      # Normal text
    ("   Leading and trailing spaces   ", 4),  # Multiple spaces
    ("Hello, world!", 2),                    # Punctuation
    ("", 0),                                 # Empty string
    ("One", 1),                              # Single word
    ("Multiple     spaces here", 3),         # Multiple spaces between words
    ("Punctuation! Does it work?", 4),       # Punctuation with multiple words
    ("123 456", 2),                          # Numbers as words
    ("Special-characters: test-case.", 3),   # Hyphens and colons
    ("   ", 0),                              # String with only spaces
]

for text, expected in test_cases:
    result = count_words(text)
    print(f"count_words({text!r}) = {result} (Expected: {expected}) - {'PASS' if result == expected else 'FAIL'}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
count_words('Hello, world!') = 2 (Expected: 2) - PASS
count_words('') = 0 (Expected: 0) - PASS
count_words('One') = 1 (Expected: 1) - PASS
count_words('Multiple     spaces here') = 3 (Expected: 3) - PASS
count_words('Punctuation! Does it work?') = 4 (Expected: 4) - PASS
count_words('123 456') = 2 (Expected: 2) - PASS
count_words('Special-characters: test-case.') = 4 (Expected: 3) - FAIL
count_words('   ') = 0 (Expected: 0) - PASS
PS C:\Users\pooji\OneDrive\Desktop\btech\2-1>
```

Task Description#4

- Generate test cases for a BankAccount class with:
  Methods:
  deposit(amount)
  withdraw(amount)
  check_balance()

Requirements:

- Negative deposits/withdrawals should raise an error.
- Cannot withdraw more than balance.

lab22.py > ...

```python
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        if amount <= 0:
            raise ValueError("Deposit amount must be positive.")
        self.balance += amount

    def withdraw(self, amount):
        if amount <= 0:
            raise ValueError("Withdrawal amount must be positive.")
        if amount > self.balance:
            raise ValueError("Insufficient funds.")
        self.balance -= amount

    def check_balance(self):
        return self.balance

# Test cases for BankAccount
def run_bank_account_tests():
    # Test 1: Normal deposit and withdrawal
    acc = BankAccount()
    acc.deposit(100)
    assert acc.check_balance() == 100, "Deposit failed"
    acc.withdraw(50)
    assert acc.check_balance() == 50, "Withdraw failed"

    # Test 2: Negative deposit
    try:
```

```
Click to add a breakpoint posit(-10)
        print("FAIL: Negative deposit did not raise error")
    except ValueError:
        print("PASS: Negative deposit raises error")

    # Test 3: Negative withdrawal
    try:
        acc.withdraw(-20)
        print("FAIL: Negative withdrawal did not raise error")
    except ValueError:
        print("PASS: Negative withdrawal raises error")

    # Test 4: Withdraw more than balance
    try:
        acc.withdraw(100)
        print("FAIL: Over-withdrawal did not raise error")
    except ValueError:
        print("PASS: Over-withdrawal raises error")

    # Test 5: Check balance after failed operations
    assert acc.check_balance() == 50, "Balance changed after failed operations"

    # Test 6: Deposit zero
    try:
        acc.deposit(0)
        print("FAIL: Zero deposit did not raise error")
    except ValueError:
        print("PASS: Zero deposit raises error")

    # Test 7: Withdraw zero
    try:
        acc.withdraw(0)
        print("FAIL: Zero withdrawal did not raise error")
    except ValueError:
        print("PASS: Zero withdrawal raises error")

    print("ALL balance checks passed.")

run_bank_account_tests()
PASS: Negative deposit raises error
PASS: Negative withdrawal raises error
PASS: Over-withdrawal raises error
PASS: Zero deposit raises error
PASS: Zero withdrawal raises error
ALL balance checks passed.
PS C:\Users\pooji\OneDrive\Desktop\btech\2-1>
```

Expected Output#4

- AI-generated test suite with a robust class that handles all test cases.

Task Description#5

Generate test cases for is_number_palindrome(num), which checks if an integer reads the same backward.

Examples:

121 → True
123 → False
0, negative numbers → handled gracefully

Expected Output#5

- Number-based palindrome checker function validated against test cases.

ROLL NO:2403A51185
BATCH NO:09

lab8.8.py > ...

```python
def is_number_palindrome(num):
    # Handle only integers
    if not isinstance(num, int) or num < 0:
        return False
    return str(num) == str(num)[::-1]

# Test cases for is_number_palindrome
palindrome_test_cases = [
    (121, True),      # Palindrome
    (123, False),     # Not a palindrome
    (0, True),        # Single digit, palindrome
    (11, True),       # Two-digit palindrome
    (22, True),       # Two-digit palindrome
    (12321, True),    # Odd-length palindrome
    (123321, True),   # Even-length palindrome
    (10, False),      # Not a palindrome
    (-121, False),    # Negative number, not palindrome
    ("121", False),   # String input, not integer
    (None, False),    # None input
    (7, True),        # Single digit
    (1001, True),     # Palindrome with zeros
    (100, False),     # Not a palindrome
]

for num, expected in palindrome_test_cases:
    result = is_number_palindrome(num)
    print(f"is_number_palindrome({num!r}) = {result} (Expected: {expected}) - {'PASS' if result == expected else 'FAIL'}")
```

```
is_number_palindrome(7) = True (Expected: True) - PASS
is_number_palindrome(1001) = True (Expected: True) - PASS
is_number_palindrome(100) = False (Expected: False) - PASS
PS C:\Users\pooji\OneDrive\Desktop\btech\2-1>
```

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

Evaluation Criteria:

Criteria	Max Marks
Task #1	0.5
Task #2	0.5
Task #3	0.5
Task #4	0.5
Task #5	0.5
Total	**2.5 Marks**